Some Parallel Linear and Nonlinear Schwarz Methods with Applications in Computational Fluid Dynamics

by

Feng-Nan Hwang

B.S., Fu-Jen Catholic University, Taiwan, 1995

M.S., University of Colorado at Denver, 1999

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Applied Mathematics

2004

This thesis entitled:
Some Parallel Linear and Nonlinear Schwarz Methods with Applications in Computational Fluid Dynamics
written by Feng-Nan Hwang
has been approved for the Department of Applied Mathematics

Xiao-Chuan Cai

Stephen F. McCormick

Date _____

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Hwang, Feng-Nan (Ph.D., Applied Mathematics)

Some Parallel Linear and Nonlinear Schwarz Methods with Applications in Computational Fluid Dynamics

Thesis directed by Prof. Xiao-Chuan Cai

Domain decomposition methods are widely used and very powerful for solving large sparse linear and nonlinear systems of equations arising from partial differential equations (PDEs). Among different families of domain decomposition methods, we focus primarily on the class of Schwarz type methods. This dissertation proposes and tests some general techniques of linear and nonlinear preconditioning based on a Schwarz framework for two such challenging problems, namely the Stokes problem and incompressible Navier-Stokes equations in computational fluid dynamics. The two-level preconditioners comprise two parts: local additive Schwarz preconditioners, which are constructed by using the solution of discrete PDEs defined on the overlapping subdomain with some proper boundary conditions, and a global coarse preconditioner, which is defined by the solution on coarse meshes of either original discrete PDEs for the Stokes problem or the linear approximation of PDEs for incompressible Navier-Stokes equations. The two-level preconditioners are applied in conjunction with some linear or nonlinear iterative methods, such as Krylov subspace methods or Newton methods. Numerical results obtained on parallel computers show that (1) for the Stokes problem, the performance of the two-level method with a multiplicative coarse preconditioner is superior to the other two variants of additive Schwarz preconditioners; (2) for incompressible Navier-Stokes equations, the local nonlinear preconditioners make the Newton method more robust in the sense that the method converges within few iterations for a wide range of Reynolds numbers and mesh sizes, and

the linear coarse preconditioner makes the method more scalable in the sense that the number of linear iterations depends only slightly on the number of parallel processors.

Dedication

Acknowledgements

Contents

Chapter

1 Introduction 1

1.1 Summary . . . 3

1.1.1 Parallel Schwarz type preconditioners for the Stokes problem 3

1.1.2 Parallel additive Schwarz preconditioned inexact Newton algorithms for incompressible Navier-Stokes equations . . . 4

1.2 Organization of dissertation . . . 5

2 Development of Parallel Software for Schwarz Methods 7

2.1 Linear and nonlinear preconditioning . . . 8

2.2 Schwarz methods . . . 11

2.3 An overview of PETSc . . . 16

2.4 Parallel implementation of Schwarz methods using PETSc . . . 22

2.4.1 External preprocessing . . . 23

2.4.2 Internal preprocessing . . . 25

2.4.3 Solution processing . . . 31

3 Models of Incompressible Flows and Discretizations 36

3.1 Introduction . . . 36

3.2 Incompressible Navier-Stokes equations and the Stokes problem . 38

3.3 Stabilized finite element formulations . . . 39

3.4 Two benchmark problems 44
3.4.1 Lid-driven cavity problem 44
3.4.2 Backward-facing step problem 44

4 Parallel Schwarz Type Preconditioners for the Stokes Problem 47
4.1 Introduction 47
4.2 Review of previous work for the saddle point problems 49
4.3 Schwarz preconditioners 53
4.3.1 One-level additive Schwarz preconditioner 54
4.3.2 Two-level methods with a parallel coarse preconditioner . . 57
4.4 Numerical results 60
4.4.1 The effect of the overlapping size 62
4.4.2 The effect of the coarse mesh size and of inexact coarse solvers 63
4.4.3 The effect of inexact subdomain solvers 65
4.4.4 Parallel performance study 68
4.5 Concluding remarks 71

5 Parallel multilevel ASPIN algorithms for incompressible Navier-Stokes equations 72
5.1 Introduction 72
5.2 Review of inexact Newton with backtracking 75
5.2.1 Line search techniques 77
5.2.2 The approximation of Jacobian matrices using multi-colored finite differences 79
5.2.3 Convergence of inexact Newton with backtracking 82
5.2.4 A simple example with unbalanced nonlinearity 84
5.3 One-level ASPIN algorithm for incompressible Navier-Stokes equations 87

5.3.1 Nonlinear additive Schwarz preconditioning 87
5.3.2 Computing the Jacobian of the preconditioned system . . . 89
5.3.3 Details of one-level additive Schwarz preconditioned inexact Newton 92
5.4 Numerical Results: I. Comparison of one-level ASPIN and NKS . 95
5.4.1 Selection of parameters for one-level ASPIN 95
5.4.2 A Newton-Krylov-Schwarz algorithm 98
5.4.3 Test 1: Lid driven cavity flow 99
5.4.4 Test 2: Flow passing a backward-facing step 111
5.5 Two-level ASPIN for incompressible Navier-Stokes equations . . . 118
5.5.1 A parallel linear coarse component for the nonlinear preconditioner 118
5.5.2 Details of the two-level ASPIN 121
5.6 Numerical results: II. 123
5.6.1 Comparison one-level ASPIN and two-level ASPIN 123
5.6.2 Preliminary timing results 128
5.7 Concluding remarks 129

6 Conclusions and Further Work 131

Bibliography 135

Tables

Table

2.1 Preconditioned GMRES(m) [92] 10

2.2 Preconditioner package in PETSc. The preconditioner with a mark "x" in "Parallel column" indicates that PETSc provides both parallel and sequential version. Otherwise the only sequential version is available. 20

2.3 Krylov Subspace package [92] in PETSc 21

2.4 Element node matrix, which relatives the local node numbers to global node numbers. The first column of this block is the list of element numbers. Each element has four nodes. The number in the last four columns represent the global node numbers in the local nodes number ordering. 26

2.5 Coordinate matrix. The first column of this block is the list of global node numbers. Each node has an unique pair of coordinates (x, y). 27

2.6 Boundary condition matrix. The first column of this block is the list of global number. Each node has a flag "0" or "1". The flag "0" means that the node is an interior node or is imposed on the Neumann boundary condition. The flag "1" means that the node is an boundary node and the Dirichlet condition is imposed. . . . 28

2.7 List of the numbering of neighboring elements. For this cases, each element has eight neighboring elements. "-1" to indicate the missing elements. 28

2.8 List of the numbering of neighboring nodes. For this cases, each node has eight neighboring nodes. "-1" to indicate the missing nodes. 29

2.9 All components in the stage of solution processing 32

2.10 Comparison of the algebraic-based (PETSc default) and the geometric-based additive Schwarz preconditioners 34

4.1 Varying the overlapping size *ovlp*. Fine mesh size: 512×512. Coarse mesh size: 40×40. All subdomain problems are solved exactly. The tolerance ϵ_C for the coarse mesh problem is set to be 10^{-10}. 64

4.2 Varying the coarse mesh size. Fine mesh: 512×512. All subdomain problems are solved exactly. $ovlp = 1$. The tolerance ϵ_C for the coarse mesh problem is set to be 10^{-10}. 66

4.3 Varying the tolerance ϵ_C for the coarse mesh problem. Fine mesh: 512×512. Coarse mesh: 40×40. All subdomain problems are solved by exact LU, $ovlp = 1$. 66

4.4 Varying the level of ILU(k) fill-in. Fine mesh: 512×512. Coarse mesh: 40×40. All subdomain problems are solved exactly. $ovlp = 1$. The tolerance ϵ_C for the iterative coarse solver is set to be 10^{-10}. 67

4.5 Fixed mesh size scalability. 70

4.6 Fixed-subdomain-mesh-size-per-processor scalability. 70

4.7 CPU time per iteration for the coarse mesh solve. 71

5.1 Number of Newton iterations: The exact Newton method is applied for original systems and preconditioned system, starting from four different initial guesses. 85

5.2 Parameters for Choice 1 and Choice 2 99

5.3 Feasible values for s_{max} and optimal values for s_{max}. A 128×128 mesh on 16 processors for $Re = 1,000, 5,000$ and $10,000$. The intervals of feasible values for S_{max} are multiplied by the first step length $||s^{(1)}||_2$. 102

5.4 Comparison of the minimum values of $\cos(\theta_k)$ for ASPIN(1) and NKS. 107

5.5 Driven cavity problem: Different mesh sizes on 16 processors. . . . 108

5.6 Driven cavity problem: Different number of processors on a 128×128 mesh. 109

5.7 Backward-facing step problem: Comparison of NKS with three choices of forcing terms. The number of Newton iterations and the average number of GMRES iterations for different values of Reynolds number. 1200×40 elements on 5×2 and 5×4 processors. "–" indicates a failure of convergence. 112

5.8 Backward-facing step problem: Comparison of NKS with three choices of forcing terms. The number of Newton iterations and the average number of GMRES iterations for different values of Reynolds number. 600×20 and 1200×40 elements on 20 (5×4) processors. "–" indicates a failure of convergence. 112

5.9 Backward-facing step problem: Different mesh sizes on 20 (5×4) processors. 113

5.10 Backward-facing step problem: Varying the overlapping sizes on a 120×40 mesh. 114

5.11 Backward-facing step problem: Different number of processors with same mesh 1200×40. 114

5.12 Backward facing step problem: Total number of subdomain nonlinear iterations. 1200 × 40 mesh, 5 × 4 subdomains on 20 processors. 117

5.13 ASPIN(2): Varying the coarse mesh size for different values of Reynolds number. Fine mesh: 128 × 128. The number of processors n_p=16. 126

5.14 ASPIN(1) and ASPIN(2): Varying the number of processors. Fine mesh size: 128 × 128. Coarse grid size: 40 × 40. 126

5.15 ASPIN(2): Varying the fine mesh size for different values of Reynolds number. Coarse mesh size: 64×64. The number of processors $np = 64$127

5.16 ASPIN(2'): Varying the fine mesh size for different values of Reynolds number. Coarse mesh size: 64×64. The number of processor $np = 64$.127

5.17 Parallel scalability of ASPIN(2') 129

5.18 Number of iterations: Break down of each component in the ASPIN(2') 129

5.19 Timing: Break down of each component in the ASPIN(2') 130

Figures

Figure

2.1 Structure of PETSc 17

3.1 Test case 1: Driven cavity problem. 45

3.2 Test case 2: A backward-facing step problem 46

4.1 Subdomain velocity and pressure space. Bullets(•) denote pressure degrees of freedom and circles(∘) denote velocity degrees of freedom. Homogenous Dirichlet boundary conditions are imposed on the interior boundary Γ_I for both $\mathbf{u}$ and p. On Γ_D only $\mathbf{u}$ is given. 56

4.2 Speedup: $T_{n_p=4}/T_{n_p}$. 69

5.1 Contour plot of $\log(0.5||r(x)||_2^2+1)$ with $m = 5$ and the histories of Newton iterations. In the case of $x^{(0)} = (0, 2)^T$, every 50 Newton iterations is denoted by a mark. In the other cases, there is a mark for each Newton iteration. 86

5.2 Contour plot of $\log(0.5||t(x)||_2^2 + 1)$ and the histories of Newton iterations. In all cases, each Newton iteration is denoted by a mark. 86

5.3 A sample mesh partition with $ovlp = 2$ on a rectangular mesh. . . 97

5.4 Streamlines. The original (left) and preconditioned (right) nonlinear system. 100

5.5 Pressure elevations. The original (top) and preconditioned (bottom) nonlinear system. 101

5.6 History of nonlinear residuals of ASPIN(1) with different values of s_{max}. A 128×128 mesh is used on 16 processors, $Re = 10,000$. ASPIN(1) with $s_{max} = 3.0$ is terminated at the earlier iteration because of the failure of backtracking. 103

5.7 History of nonlinear residuals. NKS with three different forcing terms. Only converged cases are labelled. 105

5.8 History of nonlinear residuals. ASPIN(1) converges in all ten test cases. 106

5.9 Numbers of subdomain nonlinear iterations are required for the global nonlinear function evaluations, which are needed in solving global Jacobian systems on different subdomains. A 128 ×128 mesh is tested on 4 processors, $Re = 10,000$. 110

5.10 Backward facing step problem: subdomain numbering 115

5.11 Backward-facing step problem: Pressure contours for different values of Reynolds number. The solutions are obtained by ASPIN using a 1200×40 mesh on 5×4 processors. 116

Chapter 1

Introduction

The main objective of this dissertation is to develop fast, robust and scalable parallel algorithms and software for solving large sparse linear and nonlinear systems of equations arising from the discretization of partial differential equations (PDEs) in computational science and engineering. More specifically, it focuses on the class of multilevel domain decomposition methods with applications in computational fluid dynamics (CFD). When designing a general parallel algorithm for solving such systems of equations, especially nonlinear systems, one must take the two important issues, *robustness* and *scalability*, into account. An algorithm is called robust if the convergence of the algorithm is not too sensitive to the changes of some system parameters, for example, the initial guess for iterative methods, the mesh size, and some physical parameters, such as the Reynolds number and the Mach number in fluid dynamics. Scalability is another important issue in parallel computing [68], and it shows how an algorithm behaves as the number of processors and the size of the problem grow. This issue becomes more significant as we solve larger and larger problems with more and more processors. In the rest of this dissertation, an algorithm is referred to as *optimal* if the iteration counts and/or the computing timing is nearly independent of the number of processors and the number of unknowns.

In the past two decades, domain decomposition has been one of the most

active research areas in computational mathematics. The family of domain decomposition methods originated in 1870 from the idea of a German mathematician, H. A. Schwarz, who had proven the existence of a solution to an elliptic Dirichlet boundary problem on an irregular domain [93]. Schwarz's approach is now known as the classical Schwarz alternating algorithm. Until the 1980s due to Lions' PhD thesis [77], this approach had not been considered to be a numerical method for the solution of systems of linear equations. Nowadays, multilevel domain decomposition methods become widely used for solving large sparse linear or nonlinear systems of equations arising from PDEs.

Among different families of domain decomposition methods, we focus primarily on the class of Schwarz type methods [87, 96], which has been proven to be theoretically optimal for many types of PDE problems [13, 14, 15, 35, 36], and practically powerful for solving very large problems on computers with thousands of processors. For example, Gordon Bell prizes, which recognize outstanding achievements in high-performance computing, were awarded to Anderson *et al.*, for an unstructured mesh Schwarz method with applications in CFD [4] and to Fischer and Tufo, for a spectral element Schwarz calculation [97].

Briefly speaking, to use a Schwarz method for linear or nonlinear systems, one first partitions the computational domain into several overlapping subdomains, then constructs a one-level additive Schwarz preconditioner by summing the solutions of discretization of original PDEs defined on subdomains with appropriate boundary conditions. The one-level additive Schwarz preconditioner is easily parallelized, since each subdomain problem is independent of each other. However, its convergence rate suffers when the number of processors is large due to the lack of communication between subdomains. For a solver to be fully scalable solver, it is necessary to develop multi-level versions of Schwarz preconditioners to increase the inter-subdomain communication. For elliptic-like problems, such

multilevel methods exist [13, 14, 15, 16, 17, 18, 35, 36], but for other, more difficult problems, such as indefinite saddle point problems, highly nonsymmetric problems, or highly nonlinear problems, very little is available in the literature. Several successful multilevel techniques are proposed and tested in the dissertation, of which a short summary is given below.

1.1 Summary

1.1.1 Parallel Schwarz type preconditioners for the Stokes problem

Indefinite saddle point problems appear in many important applications, including the modeling of slow flows in fluid dynamics, isotropic incompressible materials in solid mechanics, and the optimal control of fluid flows. Such systems are highly ill-conditioned, and the design of a robust and efficient preconditioner is critical for the convergence of any iterative method. Klawonn and Pavarino [71] first applied the Schwarz method to the Stokes problem. Using a Matlab code, they showed numerically that by adding a coarse solver, the two-level additive Schwarz method is scalable with respect to the number of subdomains, although the mathematical analysis of the method is still an open problem. In Chapter 4, we focus on the parallel performance of a Stokes solver with three types of overlapping Schwarz preconditioners: a one-level additive Schwarz method, a two-level additive Schwarz method, and a two-level hybrid Schwarz method, which incorporates a coarse solver and a one-level method in a multiplicative fashion. In comparison with the two additive Schwarz methods, the hybrid method showed superior parallel performance – excellent parallel efficiency, superlinear speedup and fast convergence. We also propose a parallel iterative coarse solver based on the one-level method defined on a partitioned coarse mesh. The numerical experiment shows that the iterative *inexact* coarse solver is sufficient to retain the

optimal convergence rate of the two-level Schwarz preconditioners for the Stokes problem. Surprisingly, previous work [14] has shown that this is not true for other types of problems such as indefinite elliptic problems, and the theory for the two-level additive Schwarz method requires an exact solve on the coarse mesh.

1.1.2 Parallel additive Schwarz preconditioned inexact Newton algorithms for incompressible Navier-Stokes equations

A much more challenging problem is to solve the nonlinear system of equations arising from a finite element discretization of incompressible Navier-Stokes equations at high Reynolds number. We are particularly interested in such problems with boundary layers or singularities. For such problems most existing nonlinear iterative methods, such as Newton methods, multigrid methods, nonlinear Krylov subspace methods, continuation methods and their variants do not work well [25, 67, 73, 75, 85, 94, 98]. They are either not robust, e.g. Newton methods, or show an unacceptably slow convergence, e.g. continuation methods. In Chapter 5, we develop a new class of multilevel nonlinear preconditioning methods [61], which enhances not only the robustness of Newton methods, but also increases the parallel scalability of the algorithm. More precisely speaking, Schwarz methods are used to construct nonlinear preconditioners for Newton methods.

The additive Schwarz preconditioned inexact Newton algorithm (ASPIN) was recently introduced by Cai and Keyes [20] for nonlinear algebraic system, and the present research extended ASPIN for incompressible Navier-Stokes equations in the primitive variable form. The sparse nonlinear system is obtained by using a stabilized finite element method on two-dimensional, unstructured meshes [12, 45, 46]. The key idea of ASPIN is that we find the solution for the original system by solving a nonlinearly preconditioned system, which has the same solution as the original system, by the inexact Newton method. The preconditioner is constructed

using the multilevel nonlinear additive Schwarz method. The one-level nonlinear preconditioners are based on the solution of the Navier-Stokes equations defined on the overlapping subdomains with some proper boundary conditions. As shown numerically in Chapter 5 as well as in [20], the one-level ASPIN is more robust than the classical Newton-Krylov-Schwarz (NKS) method for high Reynolds number flows as well as for a moderate numbers of processors. However, without a coarse space, the number of iterations for solving global Jacobian systems, which is the most expensive step in the Newton type method, is not scalable with respect to the number of processors.

To improve the scalability of ASPIN, we introduce a new two-level nonlinear preconditioner, which combines a local one-level nonlinear additive Schwarz preconditioner with a global *linear* coarse preconditioner [61]. This approach is more attractive than the two-level method introduced by Cai *et al.* [21], which is *nonlinear* on both levels. Since the coarse mesh function evaluation requires only the solution of a linear coarse system rather than a nonlinear coarse system derived from the discretization of original PDEs, the overall computational cost is reduced considerably. The new two-level ASPIN retains the fast convergence and robustness properties of the one-level ASPIN. In addition, if the coarse mesh size is fine enough, the new algorithm provides better nonlinear and linear scalability with respect to the number of processors.

1.2 Organization of dissertation

This dissertation comprises six chapters. The next chapter provides an overview of Schwarz methods, including the their construction, a summary of theoretical analysis for certain classes of PDEs, as well as some issues of parallel implementation of Schwarz methods.

Chapter 3 describes two model problems in incompressible flows, the first the

linear Stokes problem, and the second the nonlinear incompressible Navier-Stokes equations. It then derives two corresponding algebraic linear and nonlinear system of equations, respectively, using stabilized finite element methods. It also includes two benchmark problems, namely a lid-driven cavity problem and backward-facing step flow problem, used for evaluating the performance of our proposed algorithms in Chapters 4 and 5.

The main contribution of this dissertation is presented in Chapters 4 and 5. Chapter 4 tests three versions of parallel Schwarz preconditioner for the Stokes problem. These Schwarz type preconditioners are a one-level additive Schwarz, a two-level additive Schwarz, and a hybrid Schwarz method, in which the coarse preconditioner is incorporated multiplicatively into the local additive Schwarz preconditioner. This chapter also presents some numerical results obtained on a parallel computer, including the study of parameter-tuning to achieve optimal parallel performance of the algorithms.

Chapter 5 proposes a new version of multi-level ASPIN for solving the incompressible Navier-Stokes equations in primitive variable form. This chapter presents a robustness comparison of one-level ASPIN and a well-understood NKS algorithm as well as a study of parallel scalability on both one-level and two-level ASPIN. Chapter 6 concludes, and also points out some possible directions for further research work.

Chapter 2

Development of Parallel Software for Schwarz Methods

In this chapter, we review a class of domain decomposition methods, namely Schwarz methods, or the overlapping domain decomposition methods, which are now among the more popular iterative methods for solving large linear systems of equations arising from PDEs in the areas of computational science and engineering. The other class of domain decomposition methods consists of non-overlapping domain decomposition methods or substructuring methods, which originated from the structural analysis community and still are widely used in solid mechanics. These two classes of domain decomposition methods differ in their ways of defining the subdomain problems related to subregions, overlapping or non-overlapping. One advantage of Schwarz methods over non-overlapping methods is that for Schwarz methods, one does not need to deal with the so-called interface problem, which may increase complexity in parallel implementation.

In general, the Schwarz methods can be employed for solving large sparse linear systems of equations in two different ways, either as an iterative method by itself or as a preconditioner for another iterative method. As an iterative method, it can be viewed as a preconditioner accelerated by simple Richardson iterative methods. Typical examples are multiplicative Schwarz Richardson methods (MSR), which are extended directly from the classical Schwarz alternating algorithm. MSR are equivalent to Gauss-Seidel like methods, in which each sub-

domain problem with an updated right-hand side is solved sequentially at each iteration. However, for some types of PDE problems, the convergence of MSR is very sensitive to the change of problem parameters and in the worst cases, fails [16, 18]. On the other hand, a more popular approach is to use the Schwarz algorithm as a preconditioner. To avoid any confusion, hereafter we use the term "Schwarz methods" refer to preconditioned iterative methods with Schwarz type preconditioners. The Schwarz type conditioners have been proven and tested to be very robust and efficient for a variety of PDE problems, even for some very ill-conditioned linear systems. Here a linear system $Ax = b$ is considered ill-conditioned if the condition number of A, defined by $\kappa(A) = ||A||\ ||A^{-1}||$, is large. In general, the condition number of a matrix, deriving from the discretization of PDEs, depends on some problem parameters, such as mesh size and other physical parameters. For example, the matrix A is obtained from five-point centered finite differences for the Poisson problem on the unit square, then $\kappa(A) = O(h^{-2})$, where h is the mesh size. The matrix becomes more ill conditioned as the meshes are refined.

2.1 Linear and nonlinear preconditioning

Solving such ill-conditioned linear systems by some iterative methods, one may experience slow convergence or failure of convergence. The technique of *preconditioning* is a way to remedy such difficulties. The key idea of preconditioning is to find the solution of the original ill-conditioned system, $Ax = b$, by solving preconditioned system, $M^{-1}A = M^{-1}b$. Here M^{-1} is called a preconditioner. If $M^{-1} = A^{-1}$, we expect that with exact arithmetic a preconditioned iterative method would require only one iteration. However, the cost of forming A^{-1} explicitly is as expensive as that of solving the original linear system. Hence it is sufficient to find an approximation of A^{-1}, M^{-1} so that $\kappa(M^{-1}A)$ is bounded

by some constants, which are independent of both the fine mesh and the number of processors, as well as of the inverse of the subdomain size, an important factor in parallel processing. In practice, there is no need to form $M^{-1}A$ explicitly. Only the computation of matrix-vector products, $u = M^{-1}Av$, is needed, if one applies Krylov subspace methods, such as the restarted generalized minimal residual (GMRES) method in Table 2.1, to solving the linear system. This is equivalent to performing $w = Av$ first, then solving $Mu = w$. Since this operation is required at each iteration (Step 3 in Table 2.1), we need to keep the cost of both the construction of M and the solution of $Mu = w$ as low as possible.

The concept of preconditioning can be generalized to the solution of ill-conditioned nonlinear systems of equations. In contrast to a common approach to dealing with nonlinear systems that involves first linearizing a nonlinear system, then solving the corresponding linearized system with preconditioning technique mentioned before, we refer to our new approach as a "*nonlinear preconditioning*" approach. The key idea of nonlinear preconditioning is similar to that of linear preconditioning. We find the solution $x = x^* \in R^n$ of the original nonlinear system $F(x) = 0$ by solving the new preconditioned nonlinear system $\mathcal{F}(x) = 0$, where $\mathcal{F}(x)$ is referred to a preconditioned nonlinear function $F(x)$. If we write $\mathcal{F}(x)$ in the form of a composite function, i.e. $\mathcal{F}(x^*) \equiv G(F(x^*)) = 0$, then $G(x)$ can be viewed as a nonlinear preconditioner for $F(x) = 0$. As in linear preconditioning, we also require some desired properties for the nonlinear preconditioner $G(x)$ as suggested by [20]:

(1) $G(0) = 0$. So this implies that if x^* is a solution of $F(x) = 0$ then x^* is also a solution of $\mathcal{F}(x) = 0$.

(2) $G(x) \approx F^{-1}(x)$ in some sense.

(3) The function evaluation $G(F(x))$ is easily computable for given $x \in R^n$.

Table 2.1: Preconditioned GMRES(m) [92]

Let x_0 be an initial guess and m the number of restarts.

1. Compute $r_0 = M^{-1}(b - Ax_0), \beta = ||r_0||_2$ and $v_1 = r_0/\beta$
2. For $j = 1, \ldots, m$ Do:
    3. Compute $w := M^{-1}Av_j$
    4. For $i = 1, \ldots, j$ Do:
    5. $h_{i,j} = (w_i, v_j)$
    6. $w := w - h_{i,j}v_i$
    7. End Do
8. Compute $h_{i+1,j} = ||w||_2$ and $v_{j+1} = w/h_{j+1,j}$
9. End Do
10. Compute $V_m := [v_1, \cdots, v_m], H_m = \{h_{i,j}\}_{1 \leq i \leq j+1; 1 \leq j \leq m}$
11. Compute $y_m = \text{argmin}_y ||\beta e_1 - H_m y||_2$, and $x_m + V_m y_m$
12. If satisfied Stop, else set $x_0 := x_m$ and GOTO 1

(4) The matrix-vector product, $(G(F(x)))'y$, is easily computable for given $x, y \in R^n$ if Newton-Krylov methods are used for solving $\mathcal{F}(x) = 0$.

The next section illustrates one way of constructing a two-level preconditioner based on the Schwarz framework. Multilevel cases can be generalized from two-level preconditioners. In general, the two-level Schwarz framework consists of four components:

(1) a set of subspaces

(2) a set of data-transfer operators between the domain and subdomains and between the fine and coarse meshes

(3) a set of subdomain problems and a coarse problem

(4) a subdomain correction operator and a coarse correction operator

2.2 Schwarz methods

Let Ω be a polygonal domain in R^d ($d = 2$ or 3) with boundary $\partial\Omega$, and consider the general linear scalar boundary value problem as follows:

$$\begin{cases} \mathcal{L}u = f \quad \text{in} \quad \Omega, \\ u = g \quad \text{on} \quad \partial\Omega \end{cases} \tag{2.1}$$

For simplicity of presentation, we take a two-dimensional Poisson problem with zero homogenous Dirichlet boundary condition, $\mathcal{L} = -\triangle$ and $g = 0$, as an example. The extension to other types of PDE problems, such as elliptic nonsymmetric and indefinite, parabolic, and hyperbolic problems, is straightforward, several related research projects having been done, including theoretical analysis and numerical investigations [13, 14, 16, 17, 18]. Some applications of Schwarz methods for solving indefinite saddle-point problems and nonlinear systems of equations are addressed in Chapters 4 and 5, respectively.

The abstract variational formulation corresponding to (2.1) is written as: Find a scalar $u \in W$ such that

$$a(u, v) = (f, v) \quad \forall v \in W, \tag{2.2}$$

where $W = H_0^1(\Omega)$ is a Hilbert space, $a(\cdot, \cdot)$ is a bilinear form from $V \times V$ to R given by $a(u, v) = \int_\Omega \nabla u \nabla v \, dx$ and $(\cdot, \cdot)$ is the scalar product in $L^2(\Omega)$ given by $(f, v) = \int_\Omega fv \, dx$. Let $\mathcal{T}^h = \{K\}$ be given a quadrilateral finite element mesh, where h is the mesh diameter defined by the longest edge of the element. Let $W^h \subset W$ be the piecewise bilinear continuous finite element space given by

$$W^h = \{v^h \in C^0(\Omega) \cap W : v|_K \in Q_1(K)^2, K \in \mathcal{T}^h\} \tag{2.3}$$

and $\{\phi_i, i = 1, \cdots n\}$ be the collection of its global basis functions.

Then the standard Galerkin finite element method is: Find $u^h \in W^h$ such that

$$a(u^h, v^h) = (f, v^h) \quad \forall v^h \in V^h. \tag{2.4}$$

When u^h is expressed in terms of the finite element basis and nodal values, i.e. $u^h = \sum_i^n c_i \phi_i$, and $v^h = \phi_j$, after the substitution of u^h and v^h into (2.4), the equivalent matrix form of (2.4) can be written as

$$Ax = b, \tag{2.5}$$

where the stiffness $A = (a_{ij})$ with $a_{ij} = a(\phi_i, \phi_j)$, the solution vector $x = (c_i)$ and the right-hand side vector $b = (b_i)$ with $b_i = (f, \phi_i)$.

To apply the Schwarz method, we begin by defining the overlapping subdomain spaces and associated subdomain problems. For subdomain partitioning, we adopt the element-based mesh partitioning, which is suitable for finite element methods. In this case, no element is split into two subdomains, so that all information related to a particular element is purely local. This is a desirable property

for distributed-memory computers, since it means that the calculation of local stiffness matrices does not involve any communications. To be more specific, let $\{\Omega_i^h, i = 1, ..., N\}$ be a non-overlapping subdomain partition with the boundary $\partial\Omega_i^h$ and assume the union of these non-overlapping subdomains covers the entire domain Ω and its mesh $\mathcal{T}^h$. We use $\mathcal{T}_i^h$ to denote the collection of mesh points in Ω_i^h. To obtain overlapping subdomains, we expand each subdomain Ω_i^h to a larger subdomain $\Omega_i^{h,\delta}$ with the boundary $\partial\Omega_i^{h,\delta}$. Here δ is an integer indicating the level of overlap. We assume that both $\partial\Omega_i^h$ and $\partial\Omega_i^{h,\delta}$ do not cut any elements of $\mathcal{T}^h$. Similarly, we use $\mathcal{T}_i^{h,\delta}$ to denote the collection of mesh points in $\Omega_i^{h,\delta}$.

Then we define the overlapping subdomain space as $W_i^h = W^h \cap H_0^1(\Omega^{h,\delta})$ and the associate subdomain problem reads: Find $u_i^h \in W_i^h$, such that

$$a(u_i^h, v_i^h) = (f, v_i^h) \qquad \forall v_i^h \in W_i^h, \tag{2.6}$$

which takes the matrix form

$$A_i x_i = b_i.$$

Now, we define the restriction operator R_i, which transfers data from $V^h \rightarrow V_i^h$. In the matrix representation, R_i is an $n_i \times n$ matrix with values of either 0 or 1, where n and n_i are the total numbers of *interior* mesh points in $\mathcal{T}^h$ and $\mathcal{T}_i^{h,\delta}$, respectively, and $\sum_{i=1}^N n_i \geq n$. Then, the interpolation operator R_i^T can be defined as the transpose of R_i.

Using restriction and interpolation operators, we can arrange subdomain corrections in additive fashion to define to the one-level additive Schwarz type preconditioner,

$$P_{ASM1}^{-1} = \sum_{i=1}^{N} R_i^T A_i^{-1} R_i. \tag{2.7}$$

Then we define the coarse-mesh part of any two-level Schwarz preconditioner, which play an important role of parallel scalability in the methods, as

follows. We assume there exists a finite element *coarse* mesh $\mathcal{T}^H$ covering the domain Ω. The two meshes $\mathcal{T}^H$ and $\mathcal{T}^h$ do not have to be nested. On the coarse mesh $\mathcal{T}^H$, we can define finite element subspaces similar to the ones defined on the fine mesh, and discretize the original PDE problem to obtain a linear system of equations,

$$A^c x^c = b^c.$$

Note that the vectors b^c and x^c are not used in the computation; only the matrix A^c is used to define our coarse preconditioner. We next define the coarse-to-fine and fine-to-coarse mesh transfer operators. Let $\{\phi_j^H(x), j = 1, \ldots, m\}$ be the finite element basis functions on the coarse mesh, and m the total number of *interior* coarse mesh points in $\mathcal{T}^H$. We define an $n \times m$ matrix I_H^h, the coarse-to-fine extension matrix, as

$$I_H^h = \left[E_1 E_2 \cdots E_n\right]^T,$$

where the block matrix E_i of size $1 \times m$ is given by

$$E_i = \begin{bmatrix} (e_H^h)_i \end{bmatrix}$$

and the row vector $(e_H^h)_i$ of length m is given by

$$(e_H^h)_i = \left[\phi_1^H(x_i), \phi_2^H(x_i), \ldots \phi_m^H(x_i)\right], \; x_i \in \mathcal{T}^h$$

for $i = 1, \ldots, n$. A global fine-to-coarse restriction operator I_h^H can be defined as the transpose of I_H^h.

Using the coarse mesh preconditioner defined above, we can define a two-level additive Schwarz preconditioner(ASM2)

$$P_{ASM2}^{-1} = I_H^h (A^c)^{-1} I_h^H + \sum_{i=1}^{N} R_i^T A_i^{-1} R_i. \tag{2.8}$$

and suggested by [78], we define the following hybrid Schwarz method(HSM),

$$P_{HSM}^{-1} = I_H^h (A^c)^{-1} I_h^H + \left(I - I_H^h (A^c)^{-1} I_h^H A\right) \left(\sum_{i=1}^{N} R_i^T A_i^{-1} R_i\right), \tag{2.9}$$

which is additive among all fine mesh subdomains, and multiplicative between the fine and coarse preconditioners. Some other two-level Schwarz preconditioner are available in literatures [15].

We state the theoretical convergence analysis of the additive Schwarz methods due to Dryja and Widlund [35, 36] without proof. Suppose that the A is a symmetric positive definite (SPD) matrix arising from the discretization of the elliptical problem and P_{ASM1}^{-1} described in (2.5) and P_{ASM2}^{-1} are one-level and two-level additive Schwarz preconditioners defined in (2.7). Assume that each subdomain problem is solved exactly. Then the condition number of $P_{ASM1}^{-1}A$ is satisfies

$$\kappa(P_{ASM1}^{-1}A) \leq \frac{C}{H^2}(1 + H/\delta),$$

and the condition number of $P_{ASM2}^{-1}A$ satisfies

$$\kappa(P_{ASM2}^{-1}A) \leq C(1 + H/\delta),$$

where the constant C is independent of the parameters H,h, and δ.

From the above theorem, we summarize the expected convergence behavior of Schwarz methods for solving the SPD problems as follows:

(1) The number of iterations for one-level methods increases proportional to $1/H$.

(2) Assume H is kept proportional to δ. For one-level methods, the number of iterations is independent of h, but not H. For two-level methods, the number of iterations is totally independent of both h and H.

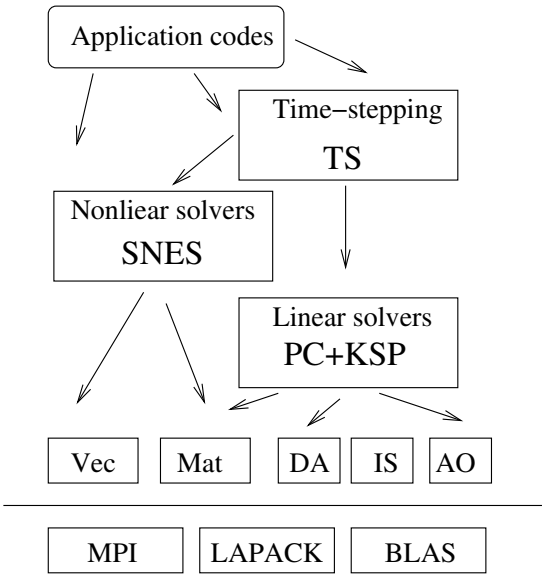
(3) For both Schwarz methods, convergence rates are very poor when $\delta = 0$ due to the condition number of $\kappa(P^{-1}A)$ is unbounded. However, the convergence rates can be improved by increasing the size of the overlap.

The analysis of the Schwarz methods has been extended to the other elliptic type problems, such as elliptic nonsymmetric indefinite problem, e.g. the convective-diffusive equation and Helmholtz equation. The condition number estimates for this type of problems are similar to that for SPD problems under an additional assumption, that the coarse problem is sufficiently fine. In practice, some requirements in the theorem can be relaxed. For example, exact subdomain solve can be replaced be inexact subdomain solve, and Schwarz methods with minimum overlap often produce the fastest convergence in terms of CPU time for many applications. However, for analysis of the strong elliptic problems we do require exact coarse solve, and numerical results are often unsatisfactory when inexact coarse solve is used.

2.3 An overview of PETSc

Our parallel implementation primarily uses a collection of libraries, the Portable, Extensible Toolkits for Scientific Computation (PETSc) [6] of the Argonne National Laboratory. The PETSc is designed for users who develop scientific codes written in FORTRAN, C or C++ programming language for solving large linear and nonlinear systems arising from PDEs with large-scale scientific applications. Several scientific PETSc-based codes have been developed for applications in a wide range of scientific fields, including nano-simulations, biology, optimization, fusion, and CFD. Those research projects represent the modern trend of development in the area of scientific computing. A complete list of related publications can be found at http://www.mcs.anl.gov/petsc. PETSc is widely available and can be downloaded at http://www.mcs.anl.gov/petsc. Technical support is also provided by the PETSc research team. PETSc offers two important features:

Figure 2.1: Structure of PETSc



- Portability: PETSc has been tested on computers system with a variety of architectures, including Dec Alpha, IBM SP, Cary T3D, etc. Users are able to test and debug codes on their local machines and easily rerun the same codes and study parallel performance on a supercomputer without any problems.

- Extensibility: In addition to using the PETSc advanced level routines for setting options and customizations, unlike most of commercial software, users are able to access the source code and modify them to meet the their special needs for developing codes based on new algorithms for different applications.

As shown in Figure 2.1, the development of PETSc adopts the principle of *software layering*. All PETSc libraries are built on top of Message Passing Interface (MPI) and two modules of linear algebra libraries: Basic Linear Algebra Subproblems (BLAS) and Linear Algebra Packages (LAPACK) library. PETSc

uses MPI Standard for distributed-memory message passing. BLAS consists of three levels of basic operations in linear algebra: vector-vector, matrix-vector, and matrix-matrix multiplications. The LAPACK library provides a variety of commonly used factorizations of the matrices, such as LU, Cholesky, QR, singular values, and Schur factorizations. The LAPACK library also provides the routines for solving linear systems of equations, least-square problems, and eigenvalue and singular value problems.

The vector (Vec) and matrix (Mat) objects are two basic objects in PETSc. One can set up some working vectors and matrices, both sequential and parallel, for solving systems of equations, such as a discrete PDE solution vector, and a right-hand side vector and coefficient matrix of a linear system of equations. These two objects also include some basic parallel vector and matrix operations, which are often used in many application codes.

In addition to the Vec and Mat objects, PETSc provides two easy-to-use data structures in conjunction with the Vec object to handle communications between the host processor and its neighboring processors. Such communications are common in parallel domain decomposition methods. For example, in the overlapping Schwarz algorithm each processor needs to collect some data, whose amount depends on the size of the overlap, from its neighboring processor first before proceeding to solve the subdomain problem. An other example is that in the calculation of a parallel nonlinear function vector at some certain vector points, each processor also needs to collect some data, whose amount is determined by the discretization stencil, from its neighboring processor first, before proceeding to evaluate some components of the vector. In particular, one can use the distributed arrays (DA) for structured meshes, while one can use the index sets (IS) for unstructured meshes.

In many applications, physical phenomena can be described by PDEs. After discretization using finite elements, finite differences, or finite volumes, these continuous models can often be written as large, algebraic, nonlinear or linear systems of equations for the steady-state problems or systems with time derivatives for time-dependent problems. PETSc provides three classes of parallel scalable solvers for these systems: linear equation, nonlinear equation, and time-stepping ordinary differential equation (ODE) solver.

(1) For linear equation solver, PETSc uses a framework of preconditioned Krylov subspace methods [33, 92]. The library of linear solvers in PETSc consists of two major components. PC is a suite of preconditioners and KSP is a suite of Krylov subspace methods, which are usable both on sequential and parallel machines. Users have several options for both preconditioners and Krylov subspaces methods, depending on their applications. PETSc also provides an interface with several external packages, including parallel sparse direct solvers, MUMPS [3], SPOOLES [5], and SuperLU_dist [28]. A list of sparse matrix solvers available in PETSc is summarized in the Tables 2.2 and 2.3. In addition, PETSc provides a type of preconditioner, namely combining preconditioner, which allows one to construct a new preconditioner by applying two different preconditioners in a additive or multiplicative fashion. However, finding such an appropriate combination is not a trivial task, and in many cases there is no gain from using a combining preconditioner rather than a single one, except in the case of multi-level preconditioners [96], where the coarse mesh part of the preconditioner is constructed by using application-provided "Shell" preconditioners.

(2) For nonlinear equation solvers, PETSc uses a framework of Newton-like

Table 2.2: Preconditioner package in PETSc. The preconditioner with a mark "x" in "Parallel column" indicates that PETSc provides both parallel and sequential version. Otherwise the only sequential version is available.

	External packages	Parallel
1. Classical iterative methods		
a. Jacobi	–	x
b. Gauss-Seidel	–	
c. SOR or SSOR	–	
2. Block or domain decomposition		
a. Block Jacobi	–	x
b. Additive Schwarz	–	x
c. Balancing Neumann-Neumann	-	x
3. Incomplete Cholesky or LU decomposition		
PETSc		
a. ICC(k)	–	
b. ILU(k)	–	
External packages		
c. ILU(0)/ICC(0)	BlockSolve95	x
4. Cholesky or LU decomposition		
PETSc		
a. LU	–	
External packages		
b. LU/LDL$^{\mathbf{T}}$	MUMPS [3]	x
c. LU/LDL$^{\mathbf{T}}$	SPOOLES [5]	x
d. LU	SuperLU_dist [28]	x
5. Approximate inverse		
a. Sparse approximate inverse	SPAI [51]	x

Table 2.3: Krylov Subspace package [92] in PETSc

KSP
a. Richardson
b. Chebyshev
c. conjugate gradients
d. GMRES
e. Bi-CG-stab
f. transpose free QMR
g. conjugate residuals
h. conjugate gradient squared
i. bi-conjugate gradient
k. MINRES
l. flexible GMRES
m. LSQR
n. SYMMLQ
o. LGMRES
p. Conjugate gradient on the normal equations

methods with two different globalization strategies: line-search techniques and trust-region methods [29, 81]. The library of nonlinear solvers in PETSc is called Simplified Nonlinear Equations Solver (SNES). To use the SNES library, users need to provide the routine for evaluating the nonlinear function. For the calculation of the corresponding Jacobian matrices, in addition to using the analytical formulation provided, users are able to form an approximation of the Jacobian matrices with multicolored finite difference methods [26] or with external automatic-differentiation software, ADIC/ADIFOR [9]. PETSc also fully supports matrix-free methods so that Jacobian-free Newton type methods [73] can be employed, often in conjunction with Krylov subspace methods.

(3) For time-stepping ODE equation solvers, PETSc uses a framework of forward/backward Euler methods. The library of ODE solvers in PETSc, namely TS, is suitable for both time-dependent problems with a semi-discretization approach [65] and steady-state problems with a pseudo-time stepping approach [25, 67, 73]. Alternately, users can employ a external parallel ODE solvers, PVODE/CVODE, now a part of SUNDIALS [57], developed by the Lawrence Livermore National Laboratory.

2.4 Parallel implementation of Schwarz methods using PETSc

The implementation of a parallel finite-element code based on overlapping Schwarz methods starting from scratch, especially in the case of unstructured meshes on a computational domain of an arbitrary shape, could be a challenging task due to presence of several non-trivial components in the code. In general, a finite-element code comprises three main stages [89]: (1) External preprocessing, (2) Internal preprocessing, and (3) Solution processing. To illustrate how to im-

plement a finite element code based on parallel Schwarz method using PETSc, we consider the following simple Poisson problem:

$$-\triangle u = f \text{ in } \Omega = [0,1] \times [0,1],$$

with Dirichlet boundary condition

$$u = 0 \text{ on } \partial\Omega$$

and $f = 8\pi^2 \sin(2\pi x) \sin(2\pi y) - 20\pi \cos(2\pi x) sin(2\pi y)$. A piecewise bilinear Galerkin finite element method is used for the discretization. For simplicity, we consider only an uniform mesh and uniform box partitioning. However, we would like to emphasis that our approach is general and also works for the case of unstructured meshes.

2.4.1 External preprocessing

In external preprocessing, this stage includes an unstructured mesh generation and a mesh partitioning for the purpose of parallel computation. Generally these two steps are done separately from main finite element codes. Since these two topics are out of our scope, we simply assume that the input data for our parallel Schwarz finite element code is generated either from some mesh partitioning packages, such as ParMeTiS [66] or other means.

Input data Input data for finite element codes usually includes three basic sets of data: the element nodes matrix, the coordinates matrix and the boundary information matrix. Tables 2.4-2.6 show an example of typical input file for two-dimensional Poisson equation using a 5×5 uniform mesh. The first block in Table 2.4 is the element nodes matrix, which relates the local node numbers in an element to global node numbering. The first column of this block is the list

of element numbers. Each element has four nodes. The number in the last four columns represent the global node numbers in the local nodes number ordering. The second block in Table 2.5 is the coordinate matrix. The first column of this block is the list of global node numbers. Each node has an unique pair of coordinates (x, y) in double precision format. The third block in Table 2.6 is the boundary condition matrix. The first column of this block is the list of global number. Each node has a flag "0" or "1". The flag "0" means that the node is an interior node or is imposed by the Neumann boundary condition. The flag "1" means that the node is an boundary node and is imposed by the Dirichlet condition.

In addition to these three matrices, we need two more index sets for the overlapping Schwarz preconditioner: the neighboring element matrix and the neighboring node matrix as shown in Tables 2.7 and 2.8 for the example.

For the purpose of parallel processing, we also need two index sets (for example, `iscell` and `isvertex`) and one global to local mapping information, which are generated by some mesh partitioning package. The first one (`iscell`) is the list of elements on each processor and the second one (`isvertex`) is the list of nodes on each processor. These two index sets can be used for generating the index for overlapping Schwarz preconditioner. For example, we partition the domain into two non-overlapping subdomains using element-based partitioning and get two index sets as follows

iscell:

```
[0] Number of indices in (stride) set 8
 [0]   [0]   [0]   [0]   [0]   [0]   [0]   [0]
  0     1     2     3     4     5     6     7
  0     1     2     3     4     5     6     7
 [1] Number of indices in (stride) set 8
```

```
 [1]   [1]   [1]   [1]   [1]   [1]   [1]   [1]
  0     1     2     3     4     5     6     7
  8     9    10    11    12    13    14    15
```

isvertex:

```
[0] Number of indices in set 15
 [0]   [0]   [0]   [0]   [0]   [0]   [0]   [0]   [0]   [0]   [0]   [0]   [0]   [0]   [0]
  0     1     2     3     4     5     6     7     8     9    10    11    12    13    14
  0     1     2     3     4     5     6     7     8     9    10    11    12    13    14

 [1] Number of indices in set 15
 [1]   [1]   [1]   [1]   [1]   [1]   [1]   [1]   [1]   [1]   [1]   [1]   [1]   [1]   [1]
  0     1     2     3     4     5     6     7     8     9    10    11    12    13    14
 11    10    15    16    12    17    13    18    14    19    20    21    22    23    24
```

2.4.2 Internal preprocessing

In addition to some general procedures for a parallel finite-element code, including initialization of processor communication utilities, such as PETSc and MPI, loading of the input data by the master processor, and then distribution of the information to each processor, as well as allocation of memory for the problem, such as working matrices and vectors, at the stage of internal preprocessing we also need to determine the overlapping subdomains used for constructing the local matrices associated with these subdomains for given PDEs problem during the stage of solution processing.

Finding overlapping subdomains We now address ways of determining the overlapping subdomains. Recall that these local matrices are obtained from the relation, $A_i = R_i A R_i^T$, where R_i and R_i^T are the restriction and prolongation matrices, respectively. Note that these restriction/prolongation matrices can be expressed in the form of index sets. The numbers in the index set indicate the locations of "1" on the diagonals of the restriction/prolongation matrices.

Table 2.4: Element node matrix, which relatives the local node numbers to global node numbers. The first column of this block is the list of element numbers. Each element has four nodes. The number in the last four columns represent the global node numbers in the local nodes number ordering.

AOData Key: cell Length 16 Ownership: 0 8 16
AOData Segment: vertex Blocksize 4 datatype int

0:	0	1	2	3
1:	1	4	5	2
2:	4	6	7	5
3:	6	8	9	7
4:	3	2	10	11
5:	2	5	12	10
6:	5	7	13	12
7:	7	9	14	13
8:	11	10	15	16
9:	10	12	17	15
10:	12	13	18	17
11:	13	14	19	18
12:	16	15	20	21
13:	15	17	22	20
14:	17	18	23	22
15:	18	19	24	23

Table 2.5: Coordinate matrix. The first column of this block is the list of global node numbers. Each node has an unique pair of coordinates (x, y).

AOData Key: vertex Length 25 Ownership: 0 13 25
AOData Segment: values Blocksize 2 datatype double

0:	0.00e+00	0.00e+00
1:	0.00e+00	2.50e-01
2:	2.50e-01	2.50e-01
3:	2.50e-01	0.00e+00
4:	0.00e+00	5.00e-01
5:	2.50e-01	5.00e-01
6:	0.00e+00	7.50e-01
7:	2.50e-01	7.50e-01
8:	0.00e+00	1.00e+00
9:	2.50e-01	1.00e+00
10:	5.00e-01	2.50e-01
11:	5.00e-01	0.00e+00
12:	5.00e-01	5.00e-01
13:	5.00e-01	7.50e-01
14:	5.00e-01	1.00e+00
15:	7.50e-01	2.50e-01
16:	7.50e-01	0.00e+00
17:	7.50e-01	5.00e-01
18:	7.50e-01	7.50e-01
19:	7.50e-01	1.00e+00
20:	1.00e+00	2.50e-01
21:	1.00e+00	0.00e+00
22:	1.00e+00	5.00e-01
23:	1.00e+00	7.50e-01
24:	1.00e+00	1.00e+00

Table 2.6: Boundary condition matrix. The first column of this block is the list of global number. Each node has a flag “0” or “1”. The flag “0” means that the node is an interior node or is imposed on the Neumann boundary condition. The flag “1” means that the node is an boundary node and the Dirichlet condition is imposed.

AOData Segment: boundary Blocksize 1 datatype logical

0:	1	12:	0
1:	1	13:	0
2:	0	14:	1
3:	1	15:	0
4:	1	16:	1
5:	0	17:	0
6:	1	18:	0
7:	0	19:	1
8:	1	20:	1
9:	1	21:	1
10:	0	22:	1
11:	1	23:	1
		24:	1

Table 2.7: List of the numbering of neighboring elements. For this cases, each element has eight neighboring elements. ”-1” to indicate the missing elements.

AOData Key: cell Length 16 Ownership: 0 8 16
AOData Segment: cell Blocksize 8 datatype int

0:	-1	-1	1	5	4	-1	-1	-1
1:	-1	-1	2	6	5	4	0	-1
2:	-1	-1	3	7	6	5	1	-1
3:	-1	-1	-1	-1	7	6	2	-1
4:	0	1	5	9	8	-1	-1	-1
5:	1	2	6	10	9	8	4	0
6:	2	3	7	11	10	9	5	1
7:	3	-1	-1	-1	11	10	6	2
8:	4	5	9	13	12	-1	-1	-1
9:	5	6	10	14	13	12	8	4
10:	6	7	11	15	14	13	9	5
11:	7	-1	-1	-1	15	14	10	6
12:	8	9	13	-1	-1	-1	-1	-1
13:	9	10	14	-1	-1	-1	12	8
14:	10	11	15	-1	-1	-1	13	9
15:	11	-1	-1	-1	-1	-1	14	10

Table 2.8: List of the numbering of neighboring nodes. For this cases, each node has eight neighboring nodes. "-1" to indicate the missing nodes.

AOData Key: vertex Length 25 Ownership: 0 13 25
AOData Segment: vertex Blocksize 8 datatype int

0:	-1	-1	1	2	3	-1	-1	-1
1:	-1	-1	4	5	2	3	0	-1
2:	1	4	5	12	10	11	3	0
3:	0	1	2	10	11	-1	-1	-1
4:	-1	-1	6	7	5	2	1	-1
5:	4	6	7	13	12	10	2	1
6:	-1	-1	8	9	7	5	4	-1
7:	6	8	9	14	13	12	5	4
8:	-1	-1	-1	-1	9	7	6	-1
9:	8	-1	-1	-1	14	13	7	6
10:	2	5	12	17	15	16	11	3
11:	3	2	10	15	16	-1	-1	-1
12:	5	7	13	18	17	15	10	2
13:	7	9	14	19	18	17	12	5
14:	9	-1	-1	-1	19	18	13	7
15:	10	12	17	22	20	21	16	11
16:	11	10	15	20	21	-1	-1	-1
17:	12	13	18	23	22	20	15	10
18:	13	14	19	24	23	22	17	12
19:	14	-1	-1	-1	24	23	18	13
20:	15	17	22	-1	-1	-1	21	16
21:	16	15	20	-1	-1	-1	-1	-1
22:	17	18	23	-1	-1	-1	20	15
23:	18	19	24	-1	-1	-1	22	17
24:	19	-1	-1	-1	-1	-1	23	18

Hence the issue here becomes how to generate index sets for Schwarz preconditioners.

For cases of regular structured meshes, the generation of such index sets is rather simple, while it seems to be very complicate in cases of unstructured meshes. Fortunately, with the help of graph theory as suggested in [96], we can describe this problem in the following way: Given a graph G_1 imbedded with another G, compute the graph G_2 that contains G_1 and all adjacent vertices, where the vertices are either nodes or elements, as in our case. Here we introduce two different approaches to obtain the index set for the overlapping Schwarz preconditioners by using PETCs commands. The first is based on elements. Once generating the index set for each overlapping element, we can extract the number of node from each element. The other is based on nodes. We can generate the index set for overlapping nodes directly. Note that there is no difference of output between these two approaches for the linear element.

(1) Case I: Element-based approach

First, we use `AODataKeyGetNeighborsIS(ao,"cell",iscell, iscell)` to get the list of elements with overlap. The new `iscell` contains the original index list of `iscell` plus neighbor element numbers provided by Table 2.7. Then, we use `AODataSegementGetReducedIS(ao,"cell","vertex", iscell,isinvertex)` to obtain the list of nodes corresponding to the overlap elements by eliminating any repeated nodes from the list.

(2) Case II: Node-based approach

We use `AODataKeyGetNeighborsIS(ao,"vertex",isvertex, &isvertex)` to get the list of nodes with overlap. The new `isvertex` contains the original index list of `isvertex` plus neighbor node numbers provided by Table 2.8.

Note that by calling `AODataKeyGetNeighborsIS` recursively, we can generate the index set for any degrees of overlap. For example, if we want to have the overlap with degree k. For the element based approach we have

```
for( i = 0; i < k-1; i++ ) {
AODataKeyGetNeighborsIS(ao,"cell",iscell,&iscell) }
AODataSegmentGetReducedIS(ao,"cell",iscell,&isinvertex)
```

Similarly, for the node based approach, we have

```
for( i = 0; i < k-1; i++ ) {
AODataKeyGetNeighborsIS(ao,"vertex",isvertex,&isvertex)
}
```

2.4.3 Solution processing

The stage of solution processing consists of two phases: problem set up and matrix solve. Table 2.9 lists all the components needed in the this stage. As shown in the table, we can see that some of necessary components have been built in PETSc, so that with the help of PETSc, the procedure for the code implementation could be simplified.

Problem set up Recall the matrix formulation of Galerkin finite element for the Poisson problem is

$$[K^e]X^e = F^e,$$

where

$$\begin{aligned} K^e_{ij} &= \int_\Omega \frac{\partial \phi_i}{\partial x}\frac{\partial \phi_j}{\partial x} + \frac{\partial \phi_i}{\partial y}\frac{\partial \phi_i}{\partial y}\, dx\, dy \\ F^e_i &= \int_\Omega f\phi_i\, dx\, dy \end{aligned}$$

Let $\{\Omega^h_i, i = 1 \cdots N\}$ be non-overlapping subdomain partition as described in

Table 2.9: All components in the stage of solution processing

ASM Components	PETSc built-in	User provided
Original PDE problem		
Constructing a matrix for the original PDE problem		x
KSP		
Krylov subspace methods	x	
PC		
Subdomain problems for each overlapping subregion		
a. Construction of data	x	x
a. Construction of subdomain matrices	x	
b. Linear solve of subdomain problems	x	
Coarse mesh problem		
a. Construction of a extension matrix		x
b. Construction of a coarse mesh matrix		x
c. Linear solve of a coarse mesh problem	x	x

Section 2.2, then K_{ij}^e can be written as

$$K_{ij}^e = \sum_{i=1}^{N} \int_{\Omega_i^h} \frac{\partial \phi_i}{\partial x} \frac{\partial \phi_j}{\partial x} + \frac{\partial \phi_i}{\partial y} \frac{\partial \phi_i}{\partial y} \, dx \, dy$$

Since each integral is independent of each other, we can calculate them in parallel and store them in each processor locally. Similarly, we can the same procedure for F_i^e.

Matrix solver Once both coefficient matrix and the working vectors, such as the solution and right-hand side vectors, have been set up, users can solve the linear system of equations by choosing any combinations of a Krylov subspace method with a preconditioner listed in Tables 2.2 and 2.3. By default, in when there is only a single processor, PETSc uses a restarted GMRES with a restart number of 30 and incomplete LU with zero fill-ins as preconditioning. When there are multiple processors, PETSc uses a parallel restarted GMRES with a restart number of 30 and block Jacobi as preconditioning, which is a special case of a one-level additive Schwarz preconditioning with zero overlap. PETSc assigns one block matrix per processor, so that sequential sparse direct LU decomposition or incomplete LU with different levels of fill-ins are suitable for a subdomain solve. Some alternative Krylov subspace methods can be used, although due to the nature of variable preconditioning, flexible GMRES needs to be used as well. Some research has been done by studying the performance of an inexact subdomain solve in domain decomposition method using iterative methods.

Furthermore, one is able to use a PETSc built-in "black-box" one-level additive Schwarz preconditioner. The default overlap is set at 1. All additional overlaps are computed internally by PETSc, and therefore subdomain matrices are constructed by PETSc as well. We refer to this type of Schwarz preconditioner as an *algebraic-based* additive Schwarz preconditioner, since it ignores all

Table 2.10: Comparison of the algebraic-based (PETSc default) and the geometric-based additive Schwarz preconditioners

n_p	$\delta = 0$	1	2	3	4	5
Algebraic-based additive Schwarz						
$2 \times 1 = 2$	18	11	9	8	8	7
$2 \times 2 = 4$	26	18	15	13	12	11
$4 \times 2 = 8$	44	25	20	17	15	14
$4 \times 4 = 16$	58	39	28	27	26	18
Geometric-based additive Schwarz						
$2 \times 1 = 2$	18	4	4	4	4	4
$2 \times 2 = 4$	22	8	8	8	8	8
$4 \times 2 = 8$	30	13	14	12	11	9
$4 \times 4 = 16$	58	22	19	17	14	10

geometric information related to an original PDEs problem, such as mesh partitioning mentioned in the previous section. Such preconditioner is easy to use, so that an application programmer, a person who is more interested in physical or engineering applications rather than software development, does not need to understand the detail of Schwarz methods in order to use them. On the other hand, we can define another type of additive Schwarz preconditioner based on the index set defined during the internal preprocessing stage, which is refereed to be as a *geometric-based* Schwarz preconditioner. Once such an index set has been generated, one can use PETSc command, `PCASMSetLocalSubdomain()`, to construct the local subdomain matrices for a particular problem geometry with any degree of overlap.

Here we compare the performance of two types of overlapping Schwarz iterative methods for the Poisson problem in the unit square. An 80×80 uniform mesh is used. From Table 2.10, we find that:

(1) If we choose shortest sides of a subdomain as its diameter, H, the number of iteration for convergence grows monotonously as $1/H$ increases for both

preconditioners.

(2) The convergence of both preconditioner is poor for $\delta = 0$, but improves as the overlap is increased.

(3) Overall, geometric-based additive Schwarz preconditioner requires a lower number of iterations to converge than does an algebraic-based additive Schwarz preconditioner.

Chapter 3

Models of Incompressible Flows and Discretizations

3.1 Introduction

Navier-Stokes equations are mathematical models used to describe the motion of a Newtonian fluid and the interaction between a fluid and its surroundings. In many applications, Navier-Stokes equations can be employed by themselves with some proper boundary conditions, such as simulations of air passing through an automobile, or they can be coupled with other systems of PDEs to model complex multi-physical phenomena. For example, the simulation of blood flow in the arteries [88] is an application of fluid-structure interaction. In this dissertation, we confine our approaches to the flows, which are assumed to be two-dimensional, steady-state, viscous, and laminar in a fixed domain. The extensions of our approaches to three-dimensional, to unsteady and to coupled systems are the focus of our ongoing projects. Some related issues are addressed in the section on future work in Chapter 6.

Navier-Stokes equations have been developed in a variety of different forms, including velocity-pressure, velocity-vorticity, and stream function-vorticity form. We consider Navier-Stokes equations only in the velocity-pressure form, also known as the primitive variable form, which is the most popular formulation in the community of computational fluid dynamics for two principle reasons:

- In practice, the pressure drop and velocity field are interesting to both

engineers and physicists. The velocity and pressure can be measured by means of experiment, and experimental data used to validate the correctness of numerical codes. Other physical quantities such as stress and stream function can be calculated from the data.

- The extension of two-dimensional cases to three-dimensional cases either in the continuous levels, e.g., building a mathematical model, or in the discretized levels, e.g., designing a scheme or a solver, is straightforward, especially in dealing with complicated boundary conditions such as interface between two fluids in a two-phase fluid problem or between a fluid and a solid in a fluid-structure interaction problem.

The next section begins with a description of two-dimensional steady-state incompressible Naiver-Stokes equations in primitive variable form. The derivation of the equations can be found in textbooks of fluid mechanics, e.g. [27, 102]. In addition to the Navier-Stokes equations, we also consider a system of PDEs, namely the Stokes problem, which models a very slow flow or a flow with very large viscosity. Section 3 formulates the stabilized finite element methods for the Stokes problem as well as for the incompressible Navier-Stokes problem and derives two large corresponding algebraic systems of equations. The stabilized finite element methods are designed to overcome numerical difficulties for the standard Galerkin finite element methods due to some characteristics of incompressible flows: (1) the incompressibility condition restricts the choices of finite element spaces for velocity and pressure. The spurious oscillation of the Galerkin finite element solution for the pressure is often observed, if equal-degree polynomials such as $Q_1 - Q_1$ element, which are the desired combinations in the implementation, are used; (2) the boundary layer presents for high Reynolds number flows, similar to the character of convective-dominated cases in the convective-diffusive equations,

so that oscillation may spoil the velocity field.

3.2 Incompressible Navier-Stokes equations and the Stokes problem

Consider two-dimensional steady-state incompressible Navier-Stokes equations in the primitive variable form [52, 89]:

$$\begin{cases} \mathbf{u}\cdot\nabla\mathbf{u} - 2\nu\nabla\cdot\epsilon(\mathbf{u}) + \nabla p = \mathbf{f} & \text{in} \quad \Omega, \\ \nabla\cdot\mathbf{u} = 0 & \text{in} \quad \Omega, \\ \mathbf{u} = \mathbf{g} & \text{on} \quad \Gamma_D, \\ \sigma\mathbf{n} = \mathbf{h} & \text{on} \quad \Gamma_N, \end{cases} \tag{3.1}$$

where $\mathbf{u} = (u_1, u_2)^T$ is the velocity, p is the pressure, ν is the dynamic viscosity, and $\epsilon(\mathbf{u}) = \frac{1}{2}[(\nabla\mathbf{u}) + (\nabla\mathbf{u})^T]$ is the symmetric part of the velocity gradient. The Cauchy stress tensor σ is defined as $\sigma = -p\mathbf{I} + 2\nu\epsilon(\mathbf{u})$, where $\mathbf{I}$ is a second-order identity tensor. For simplicity, in this dessertation the body force $\mathbf{f} = 0$. Here we assume that Ω is a bounded domain in R^2 with a polygonal boundary Γ. Two types of boundary conditions are imposed: Dirichlet conditions (Γ_D) and Neumann conditions (Γ_N). Note that if only the Dirichlet boundary is specified, i.e., $\Gamma_N = \phi$, the pressure p is determined up to a constant. To make p unique, we impose an additional condition:

$$\int_\Omega p\, dx = 0.$$

Also the Reynolds number Re, the ratio of inertia force to viscous force, can be defined as UL/ν, where U is the characteristic of flow velocity and L is characteristic of length. These two constants can be chosen arbitarily. When the Reynolds number is very small, and the effect of the viscous force plays more signifiant role than the one of the inertia force. Therefore the convectivie term $\mathbf{u}\cdot\nabla\mathbf{u}$ in (3.1) can be neglected. Using the fact that $\nabla\cdot(\nabla\mathbf{u})^T = \nabla(\nabla\cdot\mathbf{u})$ and the incompressibility

condition, $\nabla \cdot \mathbf{u} = 0$, the diffusive term $2\nu\nabla \cdot \epsilon(\mathbf{u})$ is reduced to $\nu\triangle\mathbf{u}$. Hence, the two-dimensional steady-state Stokes problem with Dirichlet boundary conditions is written as:

$$\begin{cases} -\nu\triangle\mathbf{u} + \nabla p = \mathbf{0} & \text{in} \quad \Omega, \\ \nabla \cdot \mathbf{u} = 0 & \text{in} \quad \Omega, \\ \mathbf{u} = \mathbf{g} & \text{on} \quad \Gamma_D, \\ \sigma\mathbf{n} = \mathbf{h} & \text{on} \quad \Gamma_N. \end{cases} \tag{3.2}$$

Without loss of generality, we assume $\nu = 1$.

3.3 Stabilized finite element formulations

To discretize (3.1) and (3.2), we use a class of stabilized $Q_1 - Q_1$ finite element methods. Assume $\mathcal{T}^h = \{K\}$ is a given conforming quadrilateral mesh [52]. For each element K we use h_K to denote its diameter. Let V^h and P^h be a pair of finite element spaces for the velocity and pressure, given by

$$\begin{aligned} V^h &= \{\mathbf{v} \in (C^0(\Omega) \cap H^1(\Omega))^2 : \ \mathbf{v}|_K \in Q_1(K)^2, \ K \in \mathcal{T}^h \ \} \\ P^h &= \{p \in C^0(\Omega) \cap L^2(\Omega) : \ p|_K \in Q_1(K), \ K \in \mathcal{T}^h\}. \end{aligned}$$

The weighting and trial velocity function spaces V_0^h and V_g^h are defined as follows:

$$V_0^h = \{\mathbf{v} \in V^h : \mathbf{v} = 0 \text{ on } \Gamma_D\} \text{ and } V_g^h = \{\mathbf{v} \in V^h : \ \mathbf{v} = g \text{ on } \Gamma_D\}.$$

The 2-norm of $\mathbf{v}$ is defined as $||\mathbf{v}||_2 = (\sum_{i=1}^{2} |v_i|^2)^{1/2}$.

Similarly, let the finite element space P_0^h be both the weighting and trial pressure function spaces:

$$P_0^h = \left\{ p \in P^h : \int_\Omega p \, dx = 0 \right\}.$$

First, we consider the steady-state Stokes problem. As suggested by [34], the Douglas-Wang stabilized finite element method for the steady-state Stokes

problem is given as: Find $\mathbf{u}^h \in V_g^h$ and $p^h \in P_0^h$, such that

$$B_S(\mathbf{u}^h, p^h; \mathbf{v}, q) = F_S(\mathbf{v}, q) \qquad \forall (\mathbf{v}, q) \in V_0^h \times P_0^h \tag{3.3}$$

with

$$\begin{aligned} B_S(\mathbf{u}, p; \mathbf{v}, q) = \; & (\nabla \mathbf{u}, \nabla \mathbf{v}) - (\nabla \cdot \mathbf{v}, p) - (\nabla \cdot \mathbf{u}, q) - \\ & \alpha \sum_{K \in \mathcal{T}_h} h_K^2 (-\triangle \mathbf{u} + \nabla p, \triangle \mathbf{u} + \nabla q)_K \end{aligned}$$

and

$$F_S(\mathbf{v}, q) = (\mathbf{h}, \mathbf{v})_{\Gamma_N}.$$

Remark 1 In [34], Douglas and Wang showed that this method is stable and has optimal convergence for any choices of positive stability parameter α. We use a constant of $\alpha = 1$ throughout this dissertation.

Next, we derive a linear algebraic system of equations in matrix form that is equivalent to (3.3). Let $\{\psi_l\}_{l=1}^{N_{enp}}$ be a set of global shape functions for $\mathbf{P}_0^h$ and the pair $(\{\psi_m\}_{m=1}^{\widetilde{N_{en}}}, \{\psi_m\}_{m=1}^{\widetilde{N_{en}}})^T$ be a set of global shape functions for $\mathbf{V}_0^h$, where N_{enp} is the number of pressure nodes and $\widetilde{N_{en}}$ is the number of velocity nodes. Both sets include boundary nodes. Without loss of generality, we assume for $m = 1, ..., N_{en} < \widetilde{N_{en}}$, that $\psi_m \neq \mathbf{0}$ is associated with interior nodes, which exclude the Dirichlet boundary node, and $\psi_m = \mathbf{0}$ for $\psi_m \in \Gamma_g$ and $m = N_{en} + 1, ..., \widetilde{N_{en}}$.

Then, the approximate primitive variables (u_1^h, u_2^h, p^h) in term of the global shape functions and nodal values can be written as

$$u_i^h = \sum_{m=1}^{N_{en}} \beta_m \psi_m^i + \sum_{m=N_{en}+1}^{\widetilde{N_{en}}} g_m^i \psi_m^i \qquad \text{for } i = 1, 2, \tag{3.4}$$

$$p^h = \sum_{l=1}^{N_{enp}} \alpha_l \phi_l, \tag{3.5}$$

where g_m^i is the nodal interpolate of $g_i(\mathbf{x})$ at $\mathbf{x}_m$.

By introducing three column vectors of shape functions, $\Psi = [\psi_1, ..., \psi_{N_{en}}]$, $\Phi = [\psi_1, ..., \psi_{N_{enp}}]$, and $\Psi_g = [\psi_{N_{en}+1}, ..., \widetilde{\psi_{N_{en}}}]$ for velocity, pressure, and shape

functions associated the Dirchilet boundary nodes, respectively, the equations (3.4) and (3.5) can be rewritten in the compact form as

$$u_i^h = \Psi^T U_i + \Psi_g^T G_i \text{ for } i = 1, 2, \tag{3.6}$$

$$p^h = \Phi^T P, \tag{3.7}$$

where $U_i = [\beta_1^i, ..., \beta_{Nen}^i]^T$ for $i = 1, 2$ and $P = [\alpha_1, ..., \alpha_{Nenp}]^T$ are column vectors of nodal values of velocity and pressure components, respectively, and $G_i = [g_{Nen+1}, ..., g_{\widetilde{Nen}}]^T$ are column vectors of interpolate vectors. Likewise, the velocity and pressure weighting functions (v_1^h, v_2^h, q^h) can be expressed as

$$v_i^h = \Psi^T V_i, \text{ for } i = 1, 2, \tag{3.8}$$

$$q^h = \Phi^T Q. \tag{3.9}$$

Substituting (3.6)-(3.9) into the formulation (3.3), we obtain a system of linear algebraic equations in the matrix form:

$$Ax = (L + L^\alpha)x = b, \tag{3.10}$$

where x is the vector of unknown nodal values of (u_1^h, u_2^h, p^h). In our implementation, after ordering the mesh points, we number unknown nodal values in the order of u_1^h, u_2^h, and p^h at each mesh point. The mesh points are grouped subdomain by subdomain for the purpose of parallel processing. More discussions on the subdomain partitioning are given in next chapter. Here, L arises from the standard Galerkin finite element formulation, including diffusive, convective, and incompressibility terms. L^α corresponds stabilized terms. In addition, the vector b is due to the inhomogeneous boundary condition.

Next, we consider steady-state incompressible Navier-Stokes equations. Following [46], the GLS finite element method for steady-state incompressible Navier-

Stokes equations reads: Find $\mathbf{u}^h \in V_g^h$ and $p^h \in P_0^h$, such that

$$B_N(\mathbf{u}^h, p^h; \mathbf{v}, q) = F_N(\mathbf{v}, q) \qquad \forall (\mathbf{v}, q) \in V_0^h \times P_0^h \tag{3.11}$$

with

$$B_N(\mathbf{u}, p; \mathbf{v}, q) = ((\nabla \mathbf{u}) \cdot \mathbf{u}, \mathbf{v}) + (2\nu\epsilon(\mathbf{u}), \epsilon(\mathbf{v})) - (\nabla \cdot \mathbf{v}, p) - (\nabla \cdot \mathbf{u}, q) +$$
$$\sum_{K \in \mathcal{T}_h} ((\nabla \mathbf{u}) \cdot \mathbf{u} + \nabla p - 2\nu \nabla \cdot \epsilon(\mathbf{u}), \tau((\nabla \mathbf{v}) \cdot \mathbf{v} - \nabla q - 2\nu \nabla \cdot \epsilon(\mathbf{v})))_K +$$
$$(\nabla \cdot \mathbf{u}, \delta \nabla \cdot \mathbf{v})$$

and

$$F_N(\mathbf{v}, q) = (\mathbf{h}, \mathbf{v})_{\Gamma_N}$$

We use the stability parameters δ and τ suggested in [46]. Let $Re_K(\mathbf{x}) = |\mathbf{u}(\mathbf{x})|_2 h_K/(12\nu)$ be an element Reynolds number, which distinguishes the locally convection-dominated flow ($Re_K(\mathbf{x}) \geq 1$) with the locally diffusion-dominated flow ($0 \leq Re_K(\mathbf{x}) < 1$). For convection-dominated elements, we use

$$\delta = \lambda |\mathbf{u}(\mathbf{x})|_2 h_K, \text{ and } \tau = \frac{h_K}{2|\mathbf{u}(\mathbf{x})|_2},$$

and for diffusion-dominated elements, we use

$$\delta = \frac{\lambda |\mathbf{u}(\mathbf{x})|_2^2 h_K^2}{12\nu}, \text{ and } \tau = \frac{h_K^2}{6\nu}.$$

Therefore, for convection-dominated regions, τ and δ are $O(h)$; for diffusion-dominated regions, τ and δ are $O(h^2)$. Note that for a fixed mesh, δ is a function of the velocity, and τ is a function of the velocity for convection-dominated regions, and is a constant for diffusion-dominated regions.

Remark 2 Franca and Frey [46] proved that the convergence of the GLS formulation holds for any combination of interpolation functions for velocity and pressure. In the implementation, it is more convenient to use equal-degree polynomials such as $Q_1 - Q_1$ element, which are usually ruled out in the standard Galerkin formulation because of the violation of the LBB condition.

Remark 3 For simplicity, we consider only rectangular bilinear $Q_1 - Q_1$ elements in this paper. The viscous terms in the GLS formulation $2\nu\nabla\cdot\epsilon(u^h)$ and $2\nu\nabla\cdot\epsilon(v^h)$ vanish. Therefore, the stabilized term on the left-hand side of the formulation (3.11) is reduced to

$$\sum_K((\nabla\mathbf{u})\cdot\mathbf{u}+\nabla p), \tau((\nabla\mathbf{v})\cdot\mathbf{v}-\nabla q)))_K$$

In this case, the GLS formulation and the Streamline-Upwind/Petrov-Galerkin (SUPG) formulation [12] coincide.

Substituting (3.6)-(3.9) into the GLS formulation (3.11), we obtain a system of nonlinear algebraic equations in the matrix form:

$$F(x) = (L + L^\tau + L^\delta)x + N(x)x + N^\tau(x)x - \widehat{F} = 0, \tag{3.12}$$

where x is the vector of unknown nodal values of (u_1^h, u_2^h, p^h). In our implementation, after ordering the mesh points, we number unknown nodal values in the order of u_1^h, u_2^h, and p^h at each mesh point. The mesh points are grouped subdomain by subdomain for the purpose of parallel processing. More discussion on the subdomain partitioning will be given later. Here, L and $N(x)$ represent the linear term and nonlinear terms, respectively, which arise from the standard Galerkin finite element formulation, In addition, the vector $\widehat{F}$ is due to the Neumann and inhomogeneous boundary condition.

In summary, we consider two following algebraic system of equations.

Problem 1 Find the solution $x = x^*$ to satisfy the linear algebraic system

$$Ax = b, \tag{3.13}$$

where matrix A and vector b are obtain from the discretization of the Stokes problem using the stabilized finite element method, and the matrix A is a large, sparse, and indefinite linear system.

Problem 2 Find the solution $x = x^*$ to satisfy the nonlinear algebraic system

$$F(x) = 0, \tag{3.14}$$

where $F(x)$ is obtained from the discretization of the Navier-Stokes equations using the stabilized finite element method, and often large, sparse, and highly nonlinear when the Reynolds number is high.

3.4 Two benchmark problems

In the following sections, we describe two benchmark problems used for evaluating the performance of our proposed algorithms in Chapters 4 and 5: a driven cavity problem [49] and a backward-facing step problem [47, 50] .

3.4.1 Lid-driven cavity problem

We consider the incompressible lid driven cavity flow defined on the unit square. The flow domain and boundary conditions are shown in Figure 3.1. Because the lid velocity $V = 1$, and the length of the lid $L = 1$, the Reynolds number for this problem is $1/\nu$. Since only Dirichlet boundary condition is specified for the velocity, the pressure is determined up to a constant. To make the pressure unique, we set its value at the lower right corner to zero.

3.4.2 Backward-facing step problem

We consider another benchmark problem often used to test the correctness and performance of numerical algorithms. For the equation (3.1) defined on a long channel $[0, 30]\times[-0.5, 0.5]$, no-slip conditions are imposed on the top and bottom walls, as well as the lower half of the left wall; i.e. $u_1 = u_2 = 0$. Detailed geometry and boundary condition information appears in Figure 3.2. A fully developed parabolic velocity profile is specified at the inlet boundary, which is

Figure 3.1: Test case 1: Driven cavity problem.

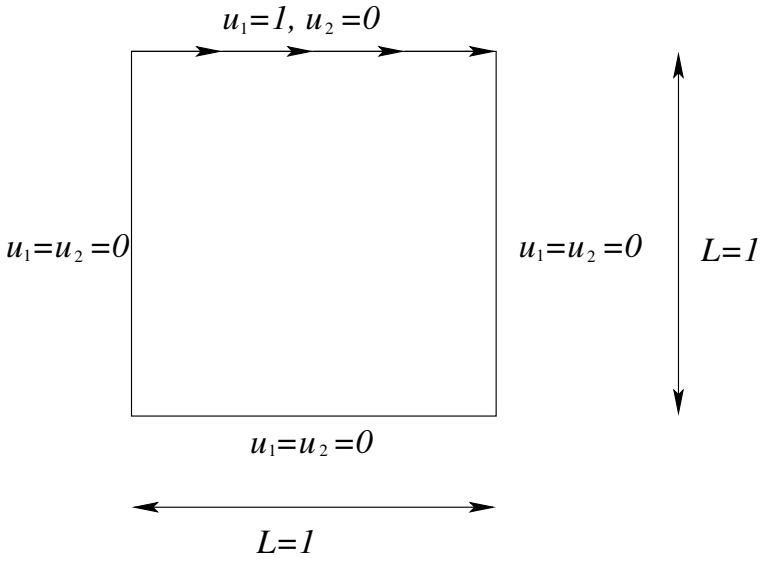
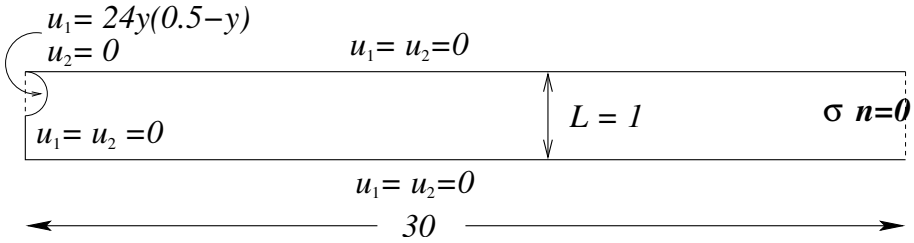


Figure 3.2: Test case 2: A backward-facing step problem



given by $u_1(y) = 24y(0.5 - y)$ for $0 \leq y \leq 0.5$. Here, we have a maximum velocity of 1.5 and an average velocity of $u_{ave} = 1$ at the inlet boundary. The Reynolds number for this problem is defined as $Re = V_{ave}L/\nu$, where V_{ave} is the average velocity at the inlet boundary (here, $V_{ave} = 1$) and L is the channel height (here, $L = 1$).

Chapter 4

Parallel Schwarz Type Preconditioners for the Stokes Problem

4.1 Introduction

The Stokes problem appears in many areas of computational science and engineering. For example, for solving incompressible Navier-Stokes equations by Newton or Picard method, the solution of Stokes or Stokes-like problems is required as a part of nonlinear iterations [43, 44, 62, 63, 64, 76] and is often the most expensive part. Another example is taken from an application of the constrained optimization problem [54, 81]. In the optimal flow control problem, by using the lagrange multiplier methods the optimization problem constrained by PDEs is reformulated as the first-order optimality system, also known as the Karush-Kuhn-Tucker (KKT) system, which is nonlinear in general. Then one needs to solve a linearized KKT system at each nonlinear iteration [7, 8, 86]. Here, the linearized KKT system has the same block structure as the Stokes problem. The Stokes problem is an indefinite saddle point problem, whose eigenvalues spread along both positive and negative real axes. As pointed out by [16, 18], unlike strong elliptic problems, two main difficulties arise in the numerical solution of indefinite linear systems:

- The lack of "good" algebraic iterative methods, such as conjugate gradient algorithm (CG) for symmetric positive definite problems, which can be

proven that the memory requirements for convergence is proportional to the number of unknowns in the linear system, but independent of the differential operators.

- The incompleteness of a mathematical theory for the convergence of the existing algebraic iterative methods, such as GMRES, which now is widely used for indefinite or nonsymmetric problems.

Consequently, the convergence of the iterative methods for indefinite linear systems is not guaranteed, and very often the convergence rate of these methods is too slow to be considered practical for large-scale applications. Therefore to resolve the situation, a good preconditioner for indefinite problems is needed. The goal of this chapter is to develop and test some scalable parallel Schwarz preconditioned iterative methods ([87, 96]) for solving the indefinite linear system (3.13), which is the discretization of the Stokes problem with the stabilized finite element method as described in Chapter 3. We solve the linear system (3.13), starting from an initial guess $x^{(0)}$, with a right preconditioned GMRES [90] method

$$AM^{-1}y = b, \text{ with } x = M^{-1}y, \tag{4.1}$$

where M^{-1} is called a right preconditioner. Other types of preconditioning, such as left or split preconditioners, could also be employed [92].

This chapter is based primarily on the paper entitled "Parallel Schwarz type preconditioners for the Stokes problem" by F. -N. Hwang and X. -C. Cai, submitted to *Computer Methods in Applied Mechanics and Engineering*, 2003 [59]. With some modifications, the remainder of this chapter is organized as follows. The next section briefly reviews some existing iterative methods for solving the saddle point problems, including the Stokes problem (3.13) and the more general Oseen problem, which consists of the linearized Navier-Stokes equations by Picard method, and then summarizes some comparisons done by other researchers

between the numerical performance of these existing methods. Section 3 introduces three types of parallel Schwarz preconditioner for the Stokes problem. This includes a one-level additive Schwarz, a two-level additive Schwarz, and a hybrid Schwarz method, in which the coarse preconditioner is incorporated multiplicatively into the local additive Schwarz preconditioner. Then, section 4 presents some numerical results obtained on a parallel computer for a lid-driven cavity flow problem, including the study of parameter-tuning to achieve optimal parallel performance of the algorithms. Finally, some remarks are presented in section 5.

4.2 Review of previous work for the saddle point problems

Several iterative methods for solving the saddle point problems are available, such as Uzawa algorithms and their variants [11, 39, 83, 105], multigrid methods [40, 62, 63, 64], Krylov subspace methods with block type preconditioners [39, 44, 69, 70, 72, 95], and non-overlapping domain decomposition methods [10, 76, 105]. Most of the existing work is based on the block structure of the discrete problem,

$$\begin{pmatrix} C & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}, \tag{4.2}$$

where C is a symmetric positive definite matrix corresponding to a diffusion operator for the Stokes problem or C is a nonsymmetric matrix corresponding to a convection-diffusion operator for the Oseen problem, and B is a matrix corresponding to a discrete divergence operator. The block structure is obtained by using an operator-splitting based ordering of the unknowns. First the velocity variables on all mesh points are ordered, followed by the pressure variables on all mesh points. Pressure projection methods, such as SIMPLE and SIMPLER methods [83], also known as Uzawa algorithms, are typical examples of the fully decoupled solution strategy. These methods decouple the full discrete system (4.2)

into two subsystems, the Schur complement system for the pressure,

$$(BC^{-1}B^T)p = BC^{-1}f - g,$$

and the Laplacean system for the velocity,

$$Cu = f - B^T p.$$

Hence, the Uzawa algorithm can be formulated as follows. First solve one of two above subsystems using some iterative methods to obtain u or p. Next determine the other variable by substituting. Then repeat the same procedure until the algorithm converges. From the two equations above, one can see that Uzawa algorithms require the product of the matrix C^{-1} with a vector at each iteration, or this is equivalent to solving the linear system $Cx = b$ with different vector b, which is the most expensive step in the algorithm. Some research work have been done to reduce the cost of the operation by replacing C by an approximation of C, such as the diagonal of C [11] or solving the linear system inexactly, such as a multigrid V-cycle or a preconditioned Krylov subspace method [40, 39].

The block structure also provides some information useful for the design of preconditioners for the saddle point problems. For example, the block diagonal preconditioner is constructed by dropping both off-diagonal blocks [40, 43, 70, 72] and the block-triangular preconditioner is obtained by replacing the lower off-diagonal block with a zero block [44, 69, 72, 95]. In general, the $(1, 1)$ block in the preconditioner is set to be the matrix C, while the common choices of the $(2, 2)$ block are the pressure Schur complement matrix $S(= BC^{-1}B^T)$, the pressure mass matrix M_p or the matrix C_p, which corresponds to the same operator as C but are defined for the pressure. As in the fully decoupled algorithm, several approaches are devoted to reducing the cost of the products of these matrices with a vector. This includes

- the use of the overlapping Schwarz preconditioners for the velocity and pressure to approximate the matrices C^{-1} and M_p^{-1}, respectively [72], or the use of a multigrid V-cycle to approximate these two block matrices [69, 70], or the use of both approaches.

- the approximation of S^{-1} by $C_p(BB^T)^{-1}$, where BB^T is a scaled discrete Laplacean, [44] or other form of approximation [95], and hence the solution of the corresponding linear system, $(BB^T)x = b$ is found by a multigrid V-cycle or preconditioned CG.

The block type preconditioners are often used in conjunction with Krylov subspace methods. In contrast to the *fully decoupled methods* mentioned above, the block preconditioned Krylov subspace methods are considered as *semi-coupled methods*, since the velocity and pressure variables are decoupled during the phase of preconditioning, i.e. the application of the preconditioner with a vector, while those these two variables are coupled together during the phase of the Krylov subspace acceleration.

On the other hand, Schwarz preconditioned methods do not require any operator-splitting or block structure. The velocity and pressure variables are always coupled together throughout the computation. One potential advantage of these *fully coupled methods* is that, without being restricted to the block structure, Schwarz type preconditioners can be applied easily to other multi-component indefinite problems, such as the flow control problem [86], which is one of the target applications of our algorithms and software. Furthermore, with small modifications, the Schwarz preconditioners studied in this chapter can be extended to solve nonsymmetric Jacobian systems in the Newton-Krylov-Schwarz algorithm for the nonlinear Navier-Stokes equations [94], and it is also possible to generalize these preconditioners for the purpose of nonlinear preconditioning, similar to the

additive Schwarz preconditioned inexact Newton algorithms [20, 21, 58]. Another class of fully coupled methods is that of multigrid method applied directly to the saddle point problem. The main difficulty of the class of these methods is how to construct an appropriate smoother in order to retain the efficiency of the multigrid method. Several smoothers have been proposed and tested [62, 63, 64], such as Vanka-type smoothers, which are equivalent to the block Gauss-Seidel methods where each block is the local saddle point problem defined on each element and Braess-Sarazin-type smoothers, which are equivalent to solving the whole system by the SIMPLE algorithm.

Since one is able to reuse the existing elliptic problem solvers and then integrate them into the computer code so that the implementation of fully decoupled approach is relatively easy, it is still one of the most popular methods in the engineering community. Several current types of commercial CFD software were developed based on the fully decoupled algorithm. For instance, CFD-ACE of CFDRC inc [106], FLOW3D of AEA Industrial Technologies [107], and Spectrum$^{\mathbf{TM}}$ of Centric Engineering System, Inc [108]. However, the convergence rate of these fully decoupled methods is very slow.

John [62] conducted a series of numerical experiments to compare the performance of the fully coupled multigrid method mentioned above with the SIMPLE algorithm for solving both of the steady-state and time-dependent incompressible Navier-Stokes equations by Picard methods. The numerical results showed that the performance of fully coupled multigrid methods is superior to that of the SIMPLE algorithm for both cases, especially the steady-state case in which the fully coupled multigrid are at least seven times faster than the SIMPLE algorithm in overall computing time. John *et al.* [64] also compared the fully coupled multigrid method with block-diagonal and block-triangular preconditioned Flexible GMRES (FGMRES) [91], where at the preconditioning stage, the linear systems are solved

by another Krylov subspace methods, e.g. GMRES or BICGSTAB [101]. They concluded that the fully coupled multigrid is the best choice among these three solvers for the incompressible Navier-Stokes equations, while the block diagonal preconditioned FGMRES fails to converges for the cases with finer meshes. In similar comparisons by Klawonn and Pavarino for the steady-state Stokes problem, the numerical results [72] showed that the overlapping additive Schwarz preconditioner is more efficient in terms of iteration counts than the block-diagonal and the block-triangular preconditioners, which diagonal blocks are constructed by overlapping Schwarz methods for the matrix C and the pressure mass matrix M_p, respectively. According these limited comparisons, the fully coupled methods seem to be a good choice for saddle point problems over other approaches such as fully decoupled and semi-coupled methods. However, in order to draw any conclusions, further numerical investigations need to be done in the future.

4.3 Schwarz preconditioners

In this section, we consider three types of Schwarz preconditioners. We begin by studying the parallel performance of the one-level additive Schwarz preconditioner proposed by Klawonn and Pavarino [71]. The preconditioner is defined by summing up the solutions of the Stokes problems on the overlapping subdomains with certain artificial boundary conditions. Although it is still not known if these boundary conditions are mathematically correct, they work well in practice. The main advantage of the one-level Schwarz preconditioner is that all subproblems are independent of each other and can be solved in parallel, but as expected, its convergence rate, in terms of the number of iterations, degenerates when the number of processors is large due to the lack of communication between subdomains. For a fully scalable method, a coarse mesh problem is required. The two-level additive Schwarz method proposed in [71, 72, 84] involves a sequential direct coarse

mesh solver, which needs to be parallelized. To have a fully scalable and fully parallel Stokes solver, we propose and test a parallel iterative coarse solver. The preconditioner for the coarse mesh problem is the same as the one-level additive Schwarz preconditioner, except that it is constructed on a partitioned coarse mesh. The coarse preconditioner can be incorporated into the fine mesh preconditioner either additively or multiplicatively. All components of the two-level method are parallel, including the coarse solver, and the restriction and interpolation operators. We show, through parallel numerical experiments, that the performance of the two-level method with a multiplicative coarse solver is superior to that of the other two variants of Schwarz preconditioners.

4.3.1 One-level additive Schwarz preconditioner

To define parallel Schwarz type preconditioners we need to partition the finite element mesh $\mathcal{T}^h$ introduced in the previous chapter. Let $\{\Omega_i^h, i = 1, ..., N\}$ be a non-overlapping subdomain partition whose union covers the entire domain Ω and its mesh $\mathcal{T}^h$. We denote the collection of mesh points in Ω_i^h by $\mathcal{T}_i^h$. To obtain overlapping subdomains, we expand each subdomain Ω_i^h to a larger subdomain $\Omega_i^{h,\delta}$ with the boundary $\partial\Omega_i^{h,\delta}$. Here δ is an integer indicating the level of overlap. We assume that $\partial\Omega_i^{h,\delta}$ does not cut any elements of $\mathcal{T}^h$. Similarly, we use $\mathcal{T}_i^{h,\delta}$ to denote the collection of mesh points in $\Omega_i^{h,\delta}$.

Now, we define the subdomain velocity space as

$$V_i^h = \{\mathbf{v}^h \in V^h \cap \left(H^1(\Omega_i^{h,\delta})\right)^2 : \mathbf{v}^h = 0 \text{ on } \partial\Omega_i^{h,\delta}\}$$

and the subdomain pressure space as

$$P_i^h = \{p^h \in P^h \cap L^2(\Omega_i^{h,\delta}) : p^h = 0 \text{ on } \partial\Omega_i^{h,\delta}\backslash\Gamma\}.$$

Both are subspaces of V^h and P^h, respectively, if we extend all subdomain functions to the whole domain by zero. See Figure 4.1 for an example of a sub-

domain mesh. Note that for $Q_1 - Q_1$ elements, we have three degrees of freedom per interior node, two for the velocity and one for the pressure. On the physical boundaries, we impose Dirichlet conditions according to the original equations (3.1). On the artificial boundaries, we assume both $\mathbf{u} = 0$ and $p = 0$. Similar boundary conditions were used in [71].

The subdomain problem reads as follows: Find $\mathbf{u}_i^h \in V_i^h$ and $p_i^h \in P_i^h$, such that

$$B(\mathbf{u}_i^h, p_i^h; \mathbf{v}, q) = F(\mathbf{v}, q) \qquad \forall (\mathbf{v}, q) \in V_i^h \times P_i^h, \tag{4.3}$$

which takes the matrix form

$$A_i x_i = b_i.$$

Note that the right-hand side of (4.3) is not important, since we only use the left-hand side matrix A_i to define the subdomain preconditioner. The right-hand side is not used at all in the computation.

Let $R_i : V^h \times P^h \to V_i^h \times P_i^h$ be a restriction operator, which returns all degrees of freedom (both velocity and pressure) associated with the subspace $V_i^h \times P_i^h$. R_i is an $3n_i \times 3n$ matrix with values of either 0 or 1, where n and n_i are the total number of mesh points in $\mathcal{T}^h$ and $\mathcal{T}_i^{h,\delta}$, respectively, and $\sum_{i=1}^N 3n_i \geq 3n$. Note that for $Q_1 - Q_1$ elements, we have three variables per mesh point, two for the velocity and one for the pressure. Then, the interpolation operator R_i^T can be defined as the transpose of R_i. The multiplication of R_i (and R_i^T) with a vector does not involve any arithmetic operations, but does involve communication in a distributed memory implementation. Using the restriction matrix, we write the one level additive Schwarz preconditioner(ASM1) in the matrix form as

$$P_{ASM1}^{-1} = \sum_{i=1}^{N} R_i^T A_i^{-1} R_i,$$

where A_i^{-1} is the subspace inverse of A_i. We remark that the global-to-local restriction operator R_i collects the data from neighboring subdomains, and the

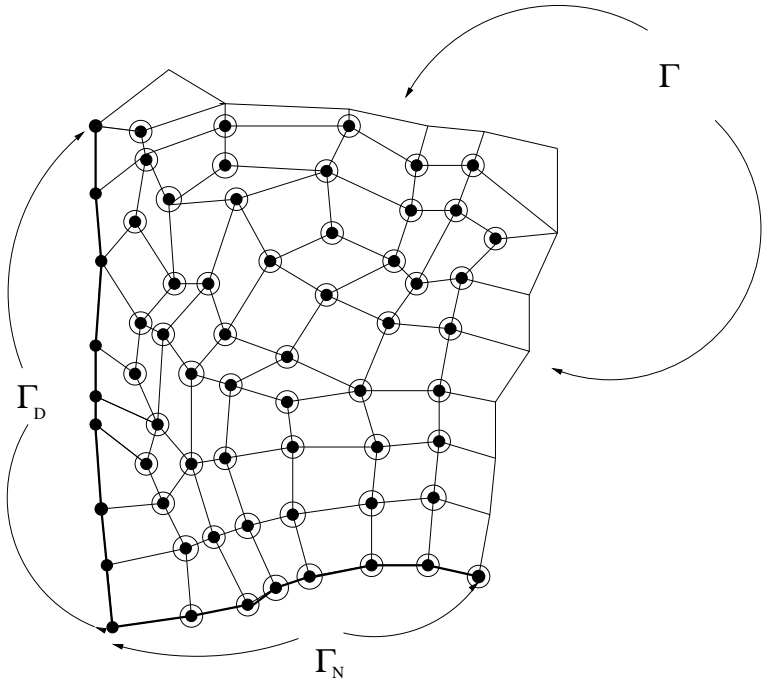


Figure 4.1: Subdomain velocity and pressure space. Bullets(•) denote pressure degrees of freedom and circles(∘) denote velocity degrees of freedom. Homogenous Dirichlet boundary conditions are imposed on the interior boundary Γ_I for both $\mathbf{u}$ and p. On Γ_D only $\mathbf{u}$ is given.

local-to-global prolongation operator R_i^T sends partial solutions to neighboring subdomains.

4.3.2 Two-level methods with a parallel coarse preconditioner

The direct coarse preconditioner proposed in [71] provides good mathematical properties, but is not easy to parallelize. Here we propose a parallel coarse problem solver, which is the same as the one-level additive Schwarz preconditioned GMRES discussed in the previous subsection, except that it is constructed on a partitioned coarse mesh. More precisely, we assume there exists a finite element mesh $\mathcal{T}^H$ covering the domain Ω. The two meshes $\mathcal{T}^H$ and $\mathcal{T}^h$ do not have to be nested. The coarse mesh problem will help speed up the convergence of the iterative method, but has nothing to do with the accuracy of the discretization, which is determined by the fine mesh $\mathcal{T}^h$ only.

For the purpose of parallel computing, the coarse mesh has to be partitioned into non-overlapping subdomains $\{\Omega_i^H\}$ and overlapping subdomains $\{\Omega_i^{H,\delta}\}$. The corresponding sets of mesh points are denoted by $\{\mathcal{T}_i^H\}$, and $\{\mathcal{T}_i^{H,\delta}\}$. For the simplicity of our software implementation, we assume that we have a *nested* non-overlapping partition. In other words, we must have

$$\Omega_i^h = \Omega_i^H$$

for $i = 1, \ldots, N$, even though the corresponding sets of mesh points do not have to be nested, i.e.,

$$\mathcal{T}_i^h \neq \mathcal{T}_i^H.$$

This also means that the same number of processors is used for both the fine and coarse mesh problems. If the overlap is taken into account, in general,

$$\Omega_i^{h,\delta} \neq \Omega_i^{H,\delta}, \quad \text{and} \quad \mathcal{T}_i^{h,\delta} \neq \mathcal{T}_i^{H,\delta}.$$

As in the fine mesh case, we can also define the restriction and extension operators R_i^c and $(R_i^c)^T$ for each coarse subdomain. On the coarse mesh $\mathcal{T}^H$, we can define finite element subspaces similar to the ones defined on the fine mesh, and discretize the original Stokes problem to obtain a linear system of equations,

$$A^c x^c = b^c.$$

Note that the vectors b^c and x^c are not used in the computation; only the matrix A^c is used to define our coarse preconditioner. Following a similar argument, on the coarse subdomains, we obtain the coarse submatrices,

$$A_i^c, i = 1, \ldots, N.$$

Unlike strong elliptic problems, our experiments show that the coarse mesh size for indefinite problems needs to be sufficiently fine to retain the scalability of the algorithm [14]. Hence, a sequential direct solver is often not feasible for such a large coarse mesh problem, especially in 3D, and therefore the coarse mesh problems need to be solved in parallel. Three strategies are applicable for the parallelization of coarse mesh solvers:

(1) **A direct exact approach**: Perform an LU decomposition of A^c and do the forward/backward substitutions in parallel using some parallel sparse direct solvers, such as SuperLU_dist [28].

(2) **A direct inexact approach**: Replace $(A^c)^{-1}$ by $\sum_{i=1}^{N}(R_i^c)^T(B_i^c)^{-1}R_i^c$, where B_i^c could be an ILU(k) decomposition of A_i^c.

(3) **An iterative approach**: Solve the coarse mesh problem, $A^c z^c = w^c$, by some parallel iterative methods, such as parallel GMRES, with a coarse additive Schwarz preconditioner.

Aitbayev *et al.* [1] and Tuminaro *et al.* [99] have studied the performance of some parallel direct exact coarse problem solvers based on the first approach and inexact domain decomposition coarse solvers based on the second approach. However, they are either too expensive or too inexact to be effective. Instead, in this paper we adopt the third approach, which is something in between the first and the second approaches, and the solution accuracy of the coarse mesh problem can be controlled easily.

Now we discuss the parallel iterative coarse solver and the operators that transfer the data between the fine mesh and the coarse mesh. We define the coarse preconditioner $(M^c)^{-1}$ in terms of a matrix-vector multiply: For any given coarse mesh vector w^c,

$$z^c = (M^c)^{-1} w^c$$

is understood as an approximate solution of the preconditioned linear system of equations,

$$A^c \left[\sum_{i=1}^{N} (R_i^c)^T (A_i^c)^{-1} R_i^c \right] y^c = w^c, \text{ with } z^c = \left[\sum_{i=1}^{N} (R_i^c)^T (A_i^c)^{-1} R_i^c \right] y^c, \tag{4.4}$$

which satisfies the condition

$$\left\| w^c - A^c z^c \right\|_2 \leq \epsilon_C \left\| w^c \right\|_2 .$$

Note that for any given w^c, the computation of z^c can be carried out in parallel using all N processors. M^c is not exactly a matrix, but a preconditioned iterative solver.

We next define the coarse-to-fine and fine-to-coarse mesh transfer operators. Let $\{\phi_j^H(x), j = 1, \ldots, m\}$ be the finite element basis functions on the coarse mesh, and m is the total number of coarse mesh points in $\mathcal{T}^H$. We define an $3n \times 3m$ matrix I_H^h, the coarse-to-fine extension matrix, as

$$I_H^h = \left[E_1 E_2 \cdots E_n \right]^T ,$$

where the block matrix E_i of size $3 \times 3m$ is given by

$$E_i = \begin{bmatrix} (e_H^h)_i & 0 & 0 \\ 0 & (e_H^h)_i & 0 \\ 0 & 0 & (e_H^h)_i \end{bmatrix}$$

and the row vector $(e_H^h)_i$ of length m is given by

$$(e_H^h)_i = \left[\phi_1^H(x_i), \phi_2^H(x_i), \ldots \phi_m^H(x_i)\right], \; x_i \in \mathcal{T}^h$$

for $i = 1, \ldots, n$. A global fine-to-coarse restriction operator I_h^H can be defined as the transpose of I_H^h.

Using the coarse mesh preconditioner defined above, we can define a two-level additive Schwarz preconditioner(ASM2)

$$P_{ASM2}^{-1} = I_H^h (M^c)^{-1} I_h^H + \sum_{i=1}^{N} R_i^T A_i^{-1} R_i. \tag{4.5}$$

Because the partitioned coarse mesh problem and the partitioned fine mesh problem are allocated to the same processor, both coarse and fine subproblems cannot be solved at the same time. Therefore it seems reasonable to consider a multiplicative approach for the coarse mesh problem in order to obtain a faster convergence. Following [78], we study the following hybrid Schwarz method(HSM),

$$P_{HSM}^{-1} = I_H^h (M^c)^{-1} I_h^H + \left(I - I_H^h (M^c)^{-1} I_h^H A\right) \left(\sum_{i=1}^{N} R_i^T A_i^{-1} R_i\right), \tag{4.6}$$

which is additive among all fine mesh subdomains, and multiplicative between the fine and coarse preconditioners.

4.4 Numerical results

In this section, we consider a two-dimensional lid-driven cavity flow problem as a benchmark for evaluating the parallel performance of the three preconditioners introduced in the previous section. Due to the nature of the inner-outer iterations in the two-level method, FGMRES [91] seems more suitable. But according

to our experiments, GMRES has no problem with convergence, therefore we use GMRES for the experiments. Several parameters in the GMRES-Schwarz algorithm need to be specified for optimal performance. In particular, we study the impact of following parameters on the convergence rate and the overall execution time: (1) the subdomain overlapping size; (2) the subdomain and coarse mesh solution quality; and (3) the coarse mesh size. We also investigate the parallel scalability, which shows how the algorithm behaves as the size of fine mesh and the number of processors grow. Since we use non-nested coarse mesh, the number of processors and the coarse mesh size are not related. We use the Portable, Extensible Toolkit for Scientific computation(PETSc) [6] for the parallel implementation and obtain all numerical results on a cluster of PCs running Linux. All timings are reported in seconds, the execution time excludes the time spent on preprocessing steps including the setup of the problem matrix, the construction of the extension matrix, and the decomposition of the meshes. The matrix factorization is included in the timing. The parameters and other details of the numerical experiments are summarized below:

- Restarted GMRES(100) is used for solving the preconditioned original linear systems (4.1) as well as the coarse mesh problem (4.4). We use a zero initial guess for both problems. The stopping criterion for (4.1) is that the condition

  $$||b - Ax^{(k)}||_2 \leq \epsilon_F ||b||_2$$

  is satisfied and we set $\epsilon_F = 10^{-5}$ for all test cases. Similarly, the stopping criterion for the coarse mesh problem (4.4) is that the condition

  $$\left\| w^c - A^c z^{(k)} \right\|_2 \leq \epsilon_C \left\| w^c \right\|_2$$

  is satisfied. Several values of ϵ_C are tested, ranging from 10^{-1} to 10^{-10}.

- Three uniform checkerboard subdomain partitions are used for our experiments: 2×2, 4×4, and 8×8. The number of subdomains is always the same as the number of processors, n_p.

- Three fine meshes are considered: 128×128, 256×256, and 512×512 and the number of unknowns ranges from 50K to 780K. The coarse mesh size varies from 16×16 to 128×128.

- The overlapping size for the fine mesh is defined as

$$ovlp = \max\left\{\frac{L'_x - L_x}{2h^K}, \frac{L'_y - L_y}{2h^K}\right\}$$

  for both interior subdomains and subdomains touching the boundary. Since square elements are used for the test problem, the elemental diameter h^Ks are the same and equal to the fine mesh size. L'_x and L'_y are defined here as the side lengths of the overlapping subdomain $\Omega_i^{h,\delta}$ in the x-direction and the y-direction, respectively. Similarly, L_x and L_y are defined as the side lengths of the non-overlapping subdomain Ω_i^h in the x-direction and the y-direction, respectively. We define the overlapping size $ovlp^H$ for the coarse mesh problem in the same fashion as above, and use the value $ovlp^H = 1$ for all test cases.

- Either a direct sparse solver or an inexact ILU(k) solver, with level of fill-in $k = 0, 1, 2, \ldots, 5$, is employed to solve the subdomain problems.

4.4.1 The effect of the overlapping size

Table 4.1 shows the effect of the overlapping size for a fixed 512×512 fine mesh with increasing number of processors, $n_p = 4, 16$, and 64. We use a fixed 40×40 coarse mesh for the two-level Schwarz preconditioners. All subdomain problems are solved by the direct sparse LU decomposition, and the tolerance for

the coarse iterative solver is set to be $\epsilon_C = 10^{-10}$. We vary the overlapping size from 1 to 16. Like elliptical-dominated type PDEs, ASM1 and ASM2 reduce the GMRES iterations monotonically as the overlapping size increases. However, the time saving is quite limited. With the optimal overlapping size the total time taken to solve the problem is 7% less than the time with the minimal overlap. On the other hand, the GMRES iterations for HSM sometimes become even larger as the overlapping size increases. The results of HSM with minimum GMRES iterations, as well as the best execution times, are obtained by using smaller overlapping size. In general, our recommendation is to use small overlap (h or $2h$) for any of the three Schwarz methods.

4.4.2 The effect of the coarse mesh size and of inexact coarse solvers

We study the effect of the coarse mesh size on the GMRES convergence and the execution time (Table 4.2). In this set of numerical experiments, a fixed 512×512 fine mesh is used. All subdomain problems are solved by an exact sparse LU decomposition and $ovlp = 1$. We vary the coarse mesh size from 16×16 to 80×80. The tolerance for the coarse iterative solver is $\epsilon_C = 10^{-10}$. The empty entry $(-)$ in the table indicates that a uniform partitioning for such coarse mesh size is not available. From Table 4.2, first, for both two-level preconditioners, when the number of processors in fixed, the number of GMRES iterations is reduced monotonically when the coarse mesh size is increased. Second, the optimal coarse mesh size for both preconditioners, based on the best execution time, is independent of the number of processors and does not need to be very fine, roughly $H \sim 15h$. Next, we relax the accuracy requirement for the coarse mesh solution, and vary $\epsilon_C = 10^{-1}$ to 10^{-10}. As shown in Table 4.3, the accuracy of the coarse solver does not need to be very high in order to retain the optimal convergence rate of the two-level Schwarz preconditioners. $\epsilon_C = 10^{-2}$ is sufficient in our experiments.

Table 4.1: Varying the overlapping size *ovlp*. Fine mesh size: 512×512. Coarse mesh size: 40×40. All subdomain problems are solved exactly. The tolerance ϵ_C for the coarse mesh problem is set to be 10^{-10}.

n_p	*ovlp*	1	2	4	8	16
			ASM1			
4	GMRES	39	36	35	33	29
	Time	95.05	**88.53**	96.48	105.32	140.26
16	GMRES	70	62	59	56	48
	Time	21.41	**19.91**	20.31	22.03	24.89
64	GMRES	168	157	129	98	77
	Time	9.31	9.28	8.62	**8.60**	9.99
			ASM2			
4	GMRES	26	24	23	21	20
	Time	87.96	**86.66**	89.31	95.65	118.27
16	GMRES	37	35	33	26	23
	Time	**16.04**	16.13	16.27	16.33	19.23
64	GMRES	43	41	36	29	25
	Time	6.21	6.19	6.03	**5.84**	7.12
			HSM			
4	GMRES	12	14	20	16	23
	Time	**71.61**	86.66	101.88	143.18	331.47
16	GMRES	19	18	30	20	31
	Time	**11.74**	11.84	15.90	15.14	22.71
64	GMRES	22	20	34	22	39
	Time	**3.67**	3.72	5.80	5.05	9.62

Surprisingly, previous work has shown that this is not true for other types of problems such as indefinite elliptic problems [14]. In [14], a theory for ASM2 requires an exact solve on the coarse mesh. Furthermore, using inexact coarse solve saves a significant amount of time in both two-level Schwarz preconditioners, especially when the number of processors is large. In the 64-processor case, an inexact solve takes only about half of the time needed for an exact solve.

4.4.3 The effect of inexact subdomain solvers

We investigate the effect of using ILU(k) as inexact subdomain solvers. In Table 4.4 we report the computational results with varying levels of fill-in in ILU(k) for a fixed 512×512 fine mesh and varying number of processors, $n_p = 4, 16$, and 64. We use a fixed 40×40 coarse mesh for the two-level Schwarz preconditioners. The tolerance for the iterative coarse solver is set to be $\epsilon_C = 10^{-10}$ and a fixed $ovlp = 1$ is used. We vary the level of fill-ins, k, from 0 to 5. Comparing Table 4.4 and Table 4.1, we observe that ASM1 with ILU(k) is not as efficient as ASM1 with exact LU. The GMRES iteration numbers become exceedingly large when the inexact solve is used. For ASM2 and HSM2, the results with optimal or nearly optimal execution time are obtained by using $k = 1$. In this case, the GMRES iteration is completely unchanged comparing with the results in Table 4.1, but the time savings for both cases are very limited as the number of processors grows. This is because solving the coarse mesh problem takes a more significant portion of the total time. We suspect the ILU(k) is not stable for the indefinite Stokes problem, and therefore do not recommend it. In the rest of the paper, we shall use exact LU for all subdomain problems.

Table 4.2: Varying the coarse mesh size. Fine mesh: 512×512. All subdomain problems are solved exactly. $ovlp = 1$. The tolerance ϵ_C for the coarse mesh problem is set to be 10^{-10}.

n_p	Coarse Mesh	16×16	20×20	32×32	40×40	64×64	80×80
ASM2							
4	GMRES	47	42	32	26	20	18
	Time	106.52	103.74	89.50	87.96	**86.88**	89.56
16	GMRES	74	61	44	37	28	24
	Time	24.73	21.21	17.26	**16.04**	16.79	18.70
64	GMRES	97	–	52	43	30	25
	Time	10.78	–	7.47	**6.21**	10.62	9.82
HSM							
4	GMRES	17	16	13	12	11	10
	Time	101.10	80.15	71.87	**71.61**	75.09	78.06
16	GMRES	29	26	21	19	14	13
	Time	13.60	12.91	12.00	**11.74**	12.15	13.44
64	GMRES	38	–	25	22	15	13
	Time	4.72	–	3.86	**3.67**	5.00	7.03

Table 4.3: Varying the tolerance ϵ_C for the coarse mesh problem. Fine mesh: 512×512. Coarse mesh: 40×40. All subdomain problems are solved by exact LU, $ovlp = 1$.

n_p	ϵ_C	1e-01	1e-02	1e-04	1e-06	1e-08	1e-10
ASM2							
4	GMRES	31	28	27	26	26	26
	Time	90.52	87.20	87.82	**83.25**	84.15	87.58
16	GMRES	40	37	37	37	37	37
	Time	15.00	**14.56**	15.46	15.59	16.33	16.06
64	GMRES	46	43	44	44	43	43
	Time	**3.47**	3.60	5.10	6.63	6.00	7.14
HSM							
4	GMRES	14	13	12	12	12	12
	Time	78.58	75.24	79.38	73.34	**71.34**	78.46
16	GMRES	24	19	19	19	19	19
	Time	11.97	**10.90**	11.37	11.50	11.73	11.76
64	GMRES	35	22	22	22	22	22
	Time	3.60	**2.39**	3.54	3.34	3.60	3.85

Table 4.4: Varying the level of ILU(k) fill-in. Fine mesh: 512×512. Coarse mesh: 40×40. All subdomain problems are solved exactly. $ovlp = 1$. The tolerance ϵ_C for the iterative coarse solver is set to be 10^{-10}.

n_p	ILU(k)	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
ASM1							
4	GMRES	3660	954	594	356	195	160
	Time	1849.26	508.34	338.65	210.87	129.26	**106.78**
16	GMRES	3895	1479	673	445	337	222
	Time	522.79	204.69	100.28	69.00	54.96	**38.27**
64	GMRES	4143	1980	962	628	491	410
	Time	138.35	69.65	36.20	24.39	20.12	**17.63**
ASM2							
4	GMRES	56	45	43	40	37	35
	Time	32.48	**27.99**	28.67	29.00	28.85	29.40
16	GMRES	56	46	45	43	41	39
	Time	10.07	**8.97**	9.36	9.15	9.34	9.46
64	GMRES	57	47	61	65	63	47
	Time	**8.44**	8.81	8.94	12.16	9.54	8.71
HSM							
4	GMRES	37	24	27	914	678	283
	Time	23.38	**17.15**	20.44	765.35	592.30	256.51
16	GMRES	38	24	27	897	496	688
	Time	10.81	**5.33**	6.27	322.50	127.37	182.51
64	GMRES	38	25	37	1503	199	158
	Time	6.64	**5.03**	7.04	233.28	33.85	24.78

4.4.4 Parallel performance study

Scalability is an important issue in parallel computing, and the issue is more significant when solving large-scale problems with many processors. To evaluate parallel scalability of the three preconditioners, we adopt the following two measures: fixed-fine-mesh-size scalability and fixed-subdomain-mesh-size-per-processor scalability. Since the problem does not fit on one processor, the parallel efficiency η is defined in the relative sense, going from n_{p1} to n_{p2} processors with $n_{p2} > n_{p1}$, as

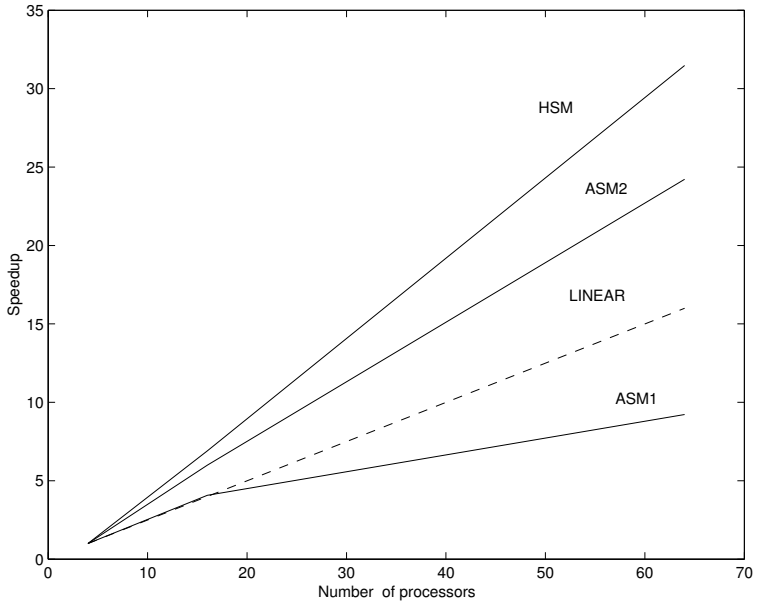
$$\eta = \frac{n_{p1} \cdot T_1}{n_{p2} \cdot T_2},$$

where T_1 and T_2 are the execution time obtained with n_{p1} and n_{p2} processors. In addition to the parallel efficiency, speedup $S = T_1/T_2$ is another useful quantity for rating the parallel performance of an algorithm. For fixed-fine-mesh-size scalability, we study how the relative parallel efficiency varies when the fixed size problem is solved by increasing number of processors. Similarly, we also study the scalability of the algorithms in the fixed-subdomain-mesh-size-per-processor sense. In the ideal situation, the algorithm takes a constant execution time if the size of problem and the number of processors increase proportionally.

In Table 4.5, we study the fixed-fine-mesh-size scalability on a fixed 512×512 mesh. The data, including the iteration numbers and the execution time for ASM1, is excerpted from the first column of Table 4.1 and for ASM2 and HSM from the second column of Table 4.3. Table 4.5 shows that, similar to elliptic dominated PDEs, the number of GMRES iterations for ASM1 grows as $\sqrt{n_p}$. HSM requires roughly half as many GMRES iterations as ASM2. Although the convergence rate shows all three preconditioners not to be scalable in terms of the iteration count, both ASM2 and HSM achieve over 100% parallel efficiency, while the parallel efficiency of ASM1 degrades about 40% as the number of processors

increases from 4 to 64. Also shown in Figure 4.2, both ASM2 and HSM have a superlinear speedup.

Figure 4.2: Speedup: $T_{n_p=4}/T_{n_p}$.



Next, we study the fixed-subdomain-mesh-size per-processor scalability. The subdomain size is fixed at 64×64. All subdomain problems are solved by the direct sparse LU decomposition with a fixed $ovlp = 1$. For two-level preconditioners, the constant ratio of fine mesh size to coarse mesh size is 8:1. The tolerance for the coarse mesh problem is set to be $\epsilon_C = 10^{-2}$. Table 4.6 shows that both two-level methods are nearly perfectly scalable in terms of the CPU time per iteration. We also see that the CPU time per iteration of ASM1 becomes even less as the number of processors increases. This might explain why the total amount of the CPU time per iteration for two-level methods remains the same, although the CPU time per iteration for solving the coarse mesh problem (Table 4.7) increases above 160% as the number of processors increases. Indeed, HSM is not only scalable but also the fastest algorithm among these three Schwarz methods under the same conditions.

Table 4.5: Fixed mesh size scalability.

n_p	GMRES	Time	Speedup	Parallel Efficiency
ASM				
4	39	86.99	1.0	1.0
16	70	21.35	4.07	1.02
64	169	9.43	9.22	0.59
ASM2				
4	26	87.20	1.0	1.0
16	37	14.56	5.99	1.49
64	43	3.60	24.22	1.51
HSM				
4	12	75.24	1.0	1.0
16	19	10.90	6.90	1.73
64	22	2.39	31.48	1.97

Table 4.6: Fixed-subdomain-mesh-size-per-processor scalability.

Mesh Size	n_p	GMRES	Time	Time per GMRES
ASM1				
128×128	4	22	1.74	0.08
256×256	16	54	3.32	0.06
512×512	64	169	9.43	0.05
ASM2				
128×128	4	24	1.99	0.08
256×256	16	29	3.51	0.12
512×512	64	30	3.14	0.10
HSM				
128×128	4	13	1.54	0.12
256×256	16	16	1.86	0.12
512×512	64	17	2.20	0.13

Table 4.7: CPU time per iteration for the coarse mesh solve.

Coarse Mesh Size	n_p	Ave. Inner GMRES	Time
ASM2			
16×16	4	6	0.004
32×32	16	10	0.012
64×64	64	21	0.028
HSM			
16×16	4	6	0.005
32×32	16	7	0.013
64×64	64	15	0.025

4.5 Concluding remarks

We have presented a fully parallel Stokes solver with three types of overlapping Schwarz preconditioners. As shown in the numerical experiments, additive Schwarz type preconditioners for the Stokes problem share some similar convergence behaviors for the elliptic-dominated equations: the number of GMRES iterations decreases monotonically as the overlapping size increases; the number of GMRES iterations for ASM1 grows as $\sqrt{n_p}$, and HSM requires roughly half as many GMRES iterations as ASM2. Our numerical results also showed that the ILU(k) based inexact subdomain solver is not useful for the Stokes problem due to the possible instability of the incomplete factorization. However, the iterative inexact coarse solver is sufficient to retain the optimal convergence rate of the two-level Schwarz preconditioners. In comparison with ASM1 and ASM2, HSM showed superior parallel performance — an excellent parallel efficiency, a super-linear speedup and a fast convergence. It would be interesting to see if the inexact coarse solver can be applied to nonsymmetric Stokes-like problems such as Oseen's equations, which are the linearized incompressible Navier-Stokes equations.

Chapter 5

Parallel multilevel ASPIN algorithms for incompressible Navier-Stokes equations

5.1 Introduction

Solving Navier-Stokes equations numerically is important, since many applications in computational science and engineering depend on it and only few cases with simple geometry have analytical solutions. Even though years of research have been spent on finding fast and reliable methods for solving Navier-Stokes equations on very fine meshes for a wide range of Reynolds numbers, it remains a difficult computing task due to certain characteristics of the equations that are yet to be fully understood mathematically, such as the boundary layer at high Reynolds numbers. To resolve the details of the solution, high resolution meshes are often required, which implies large condition numbers of the resulting algebraic systems, and also implies the need for a large-scale parallel computer. The focus of this chapter is to develop a fast, scalable, and robust parallel iterative algorithm and software for solving large, sparse, nonlinear systems of equations (3.14) in Problem 2, which is the finite element discretization of steady-state incompressible Navier-Stokes equations in the velocity-pressure formulation. We solve the nonlinear algebraic system (3.14) using the class of Newton methods with nonlinear preconditioning. The inexact Newton method (IN) is a popular technique for solving nonlinear systems, since it is easy to implement, general-purpose and

has a rapid convergence rate when the initial guess is near the desired solution [29, 30, 41]. One drawback is that a good enough initial guess is often not easily obtainable, especially for high Reynolds number steady-state flows. It is well known from numerical experience that the radius of convergence for IN becomes smaller as the Reynolds number increases. As a result, IN often diverges, even at moderate Reynolds numbers and with globalization techniques, such as line search or trust region [29, 81].

To enhance the robustness of nonlinear iterative methods, several techniques have been proposed for finding a good initial guess. For example, continuation methods are popular choices for solving high Reynolds number flow problems, see [53, 55, 56] and references therein. In general, continuation methods fall into three categories: parameter continuation [2, 53], mesh sequencing [75], and pseudo time stepping [25, 67, 73]. The advantage of continuation is that the implementation is often relatively easy and robust. On the other hand, the convergence is usually slow, and is not efficient for large scale problems. Furthermore, we do not always have the criteria for determining the size of the incremental parameter, or the optimal size of a coarse mesh, or the optimal choice of the time increment for each iteration. Alternately, to obtain the convergence of IN for high Reynolds number flow problems, Shadid *et al.* [94] introduced an algorithm based on an inexact Newton method with backtracking (INB). For theoretical discussions of INB, see [41, 42]. The key component of INB is the forcing term, which provides a criterion for determining the accuracy of the Jacobian solver during Newton iterations. Instead of using a constant forcing term throughout the computation, INB becomes more robust and efficient if the forcing term is chosen adaptively based on the nonlinear residual at the previous Newton step. The right combinations of the forcing term and the certain discretization parameters lead to nice convergence even for high Reynolds numbers. However, numerical experiments conducted by

us and others [94] have showed that the selection of the forcing terms is quite problem-dependent. In other words, the nonlinear solver may diverge because of a slight change of a problem parameter.

Recently, without employing any of the techniques discussed above, a more robust parallel algorithm, namely a nonlinear additive Schwarz preconditioned inexact Newton method (ASPIN), was introduced [20, 21, 22, 73] for solving large sparse nonlinear systems of equations arising from the discretization of nonlinear partial differential equations. ASPIN has been shown numerically to be more robust than INB for solving some challenging flow problems such as incompressible Navier-Stokes equations in the velocity-vorticity formulation [20, 22] and a full potential flow problem [21]. The key idea of ASPIN is that we find the solution of the original system $F(x) = 0$ by solving a nonlinearly preconditioned system $\mathcal{F}(x) = 0$ that has the same solution, but with more balanced nonlinearities.

In this chapter, we propose a new version of ASPIN for solving incompressible Navier-Stokes equations in the primitive variable form. The sparse nonlinear system is obtained by using a Galerkin least squares finite element (GLS) discretization [45, 46] on two dimensional unstructured meshes. A re-scaling step is added to the ASPIN algorithm in [20] to make the algorithm more suitable for the primitive variable Navier-Stokes equations, which can be used to model many different flows.

The material of this chapter is based primarily on three papers by F. -N. Hwang and X. -C. Cai: "A parallel nonlinear additive Schwarz preconditioned inexact Newton algorithm for incompressible Navier-Stokes equations", submitted to *Journal of Computational Physics*, 2003 [58]; "Improving robustness and parallel scalability of Newton's method through nonlinear preconditioning", in *the Proceedings of the 15th International Conference on Domain Decomposition Methods*, 2004 [60]; "Reducing nonlinear complexity of two-level nonlinear addi-

tive Schwarz preconditioned inexact Newton method, in preparation [61].

This chapter is organized as follows. Section 2 briefly reviews an inexact Newton method with backtracking for solving nonlinear systems of equations. Section 3 describes the details of one-level ASPIN for the incompressible Navier-Stokes equations. Section 4 presents numerical results obtained on a parallel computer for two benchmark problems: a driven cavity flow problem [49] and a backward-facing step problem [47, 50]. We compare our approach with the well-understood, Newton-Krylov-Schwarz (NKS) algorithm [19, 73, 94]. Section 5 discusses a two-level ASPIN algorithm and introduces a new linear coarse solver, which plays the central role in the scalability of the algorithm. Section 6 presents some numerical results obtained on a parallel computer for a lid-driven cavity flow problem. Particularly, it focuses on the parallel linear and nonlinear scalability of the method and include some preliminary timing results. Finally, conclusions and some remarks are given in section 7.

5.2 Review of inexact Newton with backtracking

In this section, we review an inexact Newton method with backtracking technique for solving general systems of nonlinear equations, including the description of a cubic line search technique and the construction of Jacobian matrices using a multicolored finite difference scheme. It is noted that INB also serves as the basis of our new preconditioned inexact Newton method. In the end of this section, we address some issues of the convergence of INB and use a simple example to illustrate that the robustness of INB can be enhanced through nonlinear preconditioning.

Consider a system of nonlinear equations

$$r(x) = 0, \tag{5.1}$$

where $r : R^n \to R^n$ given by $r = [r_1, r_2, \cdots, r_n]^T$. Assume $x = x^*$ is a solution of $r(x) = 0$ and its Jacobian matrices r' exist. Let $x^{(0)}$ be a given initial guess, and $x^{(k)}$ the current approximate solution. Then a new approximate solution $x^{(k+1)}$ of (5.1) can be computed via the following steps:

Algorithm 1 (Inexact Newton with Backtracking) .

Step 1: Find an inexact Newton direction $s^{(k)}$ such that

$$||r(x^{(k)}) + r'(x^{(k)})s^{(k)}||_2 \leq \eta_k ||r(x^{(k)})||_2.$$

Step 2: Compute a new approximate solution $x^{(k+1)}$ with backtracking

$$x^{(k+1)} = x^{(k)} + \lambda^{(k)} s^{(k)}.$$

In INB, the scalar η_k is often called the "forcing term", which determines how accurately the Jacobian system needs to be solved by some iterative method. A Krylov subspace type method, such as GMRES [90], is commonly used for solving the preconditioned Jacobian system,

$$M_k^{-1} r'(x^{(k)}) s^{(k)} = M_k^{-1} r(x^{(k)}), \tag{5.2}$$

where M_k^{-1} is a left preconditioner. If the chosen forcing terms are small enough, the algorithm reduces to the exact Newton algorithm. On the other hand, we can determine the forcing terms based on the information from the previous Newton iteration. Two common choices of forcing terms were suggested by Eisenstat and Walker [42]:

- Choice 1: Select any $\eta_0 \in [0, 1)$ and for $k = 1, 2, ...,$ choose

$$\eta_k = \frac{\left| ||r(x^{(k)})||_2 - ||r(x^{(k-1)}) + r'(x^{(k-1)})s^{(k-1)}||_2 \right|}{||r(x^{(k-1)})||_2}. \tag{5.3}$$

- Choice 2: Given $\gamma \in [0,1]$ and $\rho \in (1,2]$, select any $\eta_0 \in [0,1)$ and for $k = 1, 2, ...,$ choose

$$\eta_k = \gamma \left(\frac{||r(x^{(k)})||_2}{||r(x^{(k-1)})||_2} \right)^{\rho}. \tag{5.4}$$

The step length, $\lambda^{(k)} \in [\lambda_{\min}, \lambda_{\max}] \subset (0,1)$, in Step 2 of Algorithm 1 is selected so that

$$f(x^{(k)} + \lambda^{(k)} s^{(k)}) \leq f(x^{(k)}) + \alpha \lambda^{(k)} \nabla f(x^{(k)})^T s^{(k)}, \tag{5.5}$$

where the two parameters $\lambda_{\min}$ and $\lambda_{\max}$ act as safeguards, which are required for strong global convergence, the merit function $f : R^n \to R$ is defined as $||r(x)||_2^2/2$, and the parameter α is used to assure that the reduction of f is sufficient. Here, a line search technique [29] is employed to determine the step length $\lambda^{(k)}$.

5.2.1 Line search techniques

The procedure for determining an optimal value of $\lambda^{(k)}$ can be written as a one-dimensional minimization problem: Find $\lambda^{(k)}$ such that

$$\min_{\lambda^{(k)} \in (0,1)} f(x^{(k)} + \lambda^{(k)} s^{(k)})$$

However, finding the exact solution of the above optimization problem is expensive; it is also unnecessary. Instead, we introduce a simple procedure for finding an solution, which satisfies the sufficient reduction condition (5.5), with little cost by constructing a quadratic or cubic polynomial model and calculating the explicit minimum of the model repeatedly until an acceptable solution is found.

Quadratic polynomial model Let $\hat{f}(\lambda) = f(x^{(k)} + \lambda s^{(k)})$ and then $\hat{f}'(\lambda) = (\nabla f(x^{(k)} + \lambda s^{(k)}))^T s^{(k)} = r(x^{(k)} + \lambda s^{(k)})^T r'(x^{(k)} + \lambda s^{(k)}) s^{(k)}$. At the kth Newton iteration, initially we have $\hat{f}(0) = \frac{1}{2}||r(x^{(k)})||_2^2$. After solving Jacobian system, we obtain two pieces of additional information: $\hat{f}(1) = ||r(x^{(k)} + s^{(k)})||_2^2$ and

$\hat{f}'(0) = (r(x^{(k)}))^T r'(x^{(k)}) s^{(k)}$. Since $s^{(k)} = -(r'(x^{(k)}))^{-1} r(x^{(k)})$, provided that the Jacobian system is solved accurately enough, we know that $\hat{f}'(0) = -||r(x^{(k)})||_2^2 < 0$. Assume that full Newton step (i.e. $\lambda = 1$) does not satisfy the sufficient reduction condition (5.5), i.e.

$$f(x^{(k)} + s^{(k)}) > f(x^{(k)}) + \alpha(\nabla f(x^{(k)}))^T s^{(k)}$$

Using the definition of $\hat{f}$ and the fact of $(\nabla f(x^{(k)}))^T = r(x^{(k)})^T (r'(x^{(k)}))$, we rewrite above inequality in terms of $\hat{f}$ and $\hat{f}'$

$$\hat{f}(1) > \hat{f}(0) + \alpha \hat{f}'(0) > \hat{f}(0) + \hat{f}'(0),$$

since $\hat{f}'(0) < 0$ and $\alpha > 0$. Hence we conclude that $\hat{f}(1) - \hat{f}(0) - \hat{f}'(0) > 0$.

Using this set of data, $\hat{f}(0)$, $\hat{f}(1)$, and $\hat{f}'(0)$, we construct a quadratic model $p(\lambda)$, which satisfies $p(0) = \hat{f}(0)$, $p'(0) = \hat{f}'(0)$ and $p(1) = \hat{f}(1)$, as follows.

$$p_2(\lambda) = \hat{f}(0) + \hat{f}'(0)\lambda + (\hat{f}(1) - \hat{f}(0) - \hat{f}'(0))\lambda^2,$$

The first derivative of $p_2(\lambda)$ is

$$p_2'(\lambda) = \hat{f}'(0) + 2(\hat{f}(1) - \hat{f}(0) - \hat{f}'(0))\lambda$$

and p_2' has a root at $\lambda^* = \dfrac{-\hat{f}'(0)}{2(\hat{f}(1) - \hat{f}(0) - \hat{f}'(0))}$, which is a local minimum of p_2, since the second derivative of $p_2(\lambda)$,

$$p_2''(\lambda) = 2(\hat{f}(1) - \hat{f}(0) - \hat{f}'(0)) > 0$$

at $\lambda = \lambda^*$. Once new $\lambda^{(k)} = \lambda^*$ is determined, we check if the updated solution, $x^{(k)} + \lambda^* s^{(k)}$, satisfies the sufficient reduction condition (5.5). If it does, the backtracking procedure stops and the next $(k+1)$th Newton iteration proceeds. Otherwise, we backtrack again either using $\hat{f}(0)$, $\hat{f}'(0)$ and $\hat{f}(\lambda^*)$ to construct a new quadratic polynomial model or using the same of the data plus $\hat{f}(1)$ to construct a cubic polynomial model as described below. Without loss the generality,

we assume λ_1 and λ_2 are two different unsuccessful backtracking trial points with $\lambda_1 < \lambda_2$

Cubic polynomial model Here, we consider a cubic polynomial $p_3(\lambda)$, which satisfies $p_3(0) = \hat{f}(0)$, $p_3'(0) = \hat{f}'(0)$, $p_3(\lambda_1) = \hat{f}(\lambda_1)$, and $p_3(\lambda_2) = \hat{f}(\lambda_2)$ as follows.

$$p_3(\lambda) = a\lambda^3 + b\lambda^2 + \hat{f}'(0) + \hat{f}(0),$$

where

$$\begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{\lambda_1^2\lambda_2^2(\lambda_1 - \lambda_2)} \begin{pmatrix} \lambda_2^2 & -\lambda_1^2 \\ -\lambda_2^3 & \lambda_1^3 \end{pmatrix} \begin{pmatrix} \hat{f}(\lambda_1) - \hat{f}(0) - \hat{f}'(0) \\ \hat{f}(\lambda_2) - \hat{f}(0) - \hat{f}'(0) \end{pmatrix}$$

The first derivative of $p_3(\lambda)$,

$$p_3'(\lambda) = 3ax^3 + 2bx + \hat{f}'(0)$$

has a root at $\lambda^+ = \dfrac{-b \pm \sqrt{b^2 - 3a\hat{f}'(0)}}{3a}$ and the root with a plus sign is a local minimum of p_3, since the second derivative of $p_3(\lambda)$,

$$p_3''(\lambda) = 6a\lambda + 2b > 0$$

at $\lambda = \lambda^+$. Again, once $\lambda = \lambda^+$ is determined, we check if the updated solution, $x^{(k)} + \lambda^+ s^{(k)}$, satisfies the sufficient reduction condition (5.5). If it does not, we repeat the same procedure until an acceptable λ is found.

5.2.2 The approximation of Jacobian matrices using multi-colored finite differences

The Jacobian matrix r' is a key component in INB. In our implementation, we form r' as a sparse matrix using multi-colored forward finite difference method [26]. For many applications, it is often difficult or expensive to compute analytically. Some other techniques are also available for approximating the Jacobian

matrix, such as automatic differentiation and symbolic differentiation. We refer to Chapter 7 of [81] for an overview of these methods. Alternatively, a matrix-free implementation of Newton-Krylov can be used to avoid the explicit computation of r'. Interested readers may refer to [73] for a complete survey of matrix-free Newton-Krylov methods as well as their applications in computational physics.

The Jacobian matrices r' can be written explicitly as $[\partial r_i(x)/\partial x_j]_{1\le i,j\le n}$. Using the forward finite difference scheme, we approximate each partial derivative $\partial r_i/\partial x_j$ at a given point x by

$$\frac{\partial r_i}{\partial x_j}(x) \approx \frac{r_i(x+\epsilon e_j) - r_i(x)}{\epsilon}, \tag{5.6}$$

where e_j denotes the jth unit vector. One important issue in the implementation is how to choose the parameter ϵ properly. We find two sources of errors in the formulation (5.6). The first error arises from the discretization of $\partial r_i/\partial x_j$, which is $O(\epsilon)$. The other error is related to the roundoff error, which is introduced by the subtraction of two real-value functions in finite-precision arithmetic. Therefore, the second error can be expressed as $O(r(x)/\epsilon)$, since $r_i(x+\epsilon e_j) \approx r_i(x)$ for small ϵ. We may reduce the first error by making ϵ as small as possible. However, the second error becomes worse when ϵ is smaller. To select an optimal value of ϵ, we need to find a compromise between these two sources of errors. Following the discussion in [81], we adopt the fairly optimal value for most cases,

$$\epsilon = \sqrt{\mathbf{w}}, \tag{5.7}$$

where the quantity $\mathbf{w}$ is an unit roundoff. Typically, $\mathbf{w}$ is about 10^{-16} in double-precision arithmetic.

Note that the finite difference formula (5.6) can also be written as

$$\frac{\partial r}{\partial x_j}(x) \approx \frac{r(x+\epsilon e_j) - r(x)}{\epsilon}. \tag{5.8}$$

Hence, forming a full Jacobian matrix requires $n+1$ function evaluations. We calculate each column of a Jacobian matrix sequentially. This process can be very expensive and time-consuming when the problem is large.

However, we can see as follows that much of this cost can be saved if the Jacobian matrices are sparse. Let $S = \{1, 2, \cdots, n\}$ be index set and each integer correspond to each r_i. First, we label each index with a "color" according to the following rule: Indices i and j are labelled with the same colors, if the evaluations of r_i and r_j do not depend the same variable x_k. Then each non-overlapping partition $S_1, S_2, \cdots, S_Q$ of S can be defined as a collection of indices with the same colors. Hence the columns of the Jacobian matrix associated with each partition can be calculated simultaneously through

$$v = \frac{r(x+p) - r(x)}{\epsilon},$$

where $p = \epsilon(\sum_{k \in S_i} e_k)$, and the vector v contains all non-zero elements of these columns. Note that only $Q+1$ function evaluations are required to form the full Jacobian matrix and usually the number of colors Q is quite small compared to the problem size n in the case of well structured sparse matrices such as those arising from the discretization of differential equations using lower-order finite element or finite difference schemes. For example, for the $n \times n$ tri-diagonal matrices, only three colors are sufficient, independent of n. Several algorithms [26] for general sparse Jacobian matrices based on graph theory and graph coloring are available, including largest-first(LF), smallest-last(SL), and incidence-degree(ID). Coleman and Moré [26] did the a performance comparison of these three algorithms and observed that latter two algorithms are often able to generate a set of "near-optimal" colorings, whose number of colors is close to the maximum number of nonzero elements in any row.

5.2.3 Convergence of inexact Newton with backtracking

In general, nonlinear iterative methods, such as INB, are very fragile. They may converge rapidly for a well-selected set of parameters (for example, certain initial guesses, certain range of Re), but diverge if we change some of the parameters slightly. They may converge well at the beginning of the iterations, then suddenly stall for no apparent reason, which is not fully understood.

In many cases, the failures of INB are likely because the angle between the Newton direction and the steepest descent direction is too close to $\pi/2$, due to the fact that Jacobian matrices are very ill-conditioned. In that case the Newton direction becomes only a weak descent direction. As a result, only extremely small steps can be accepted by the line search. More precisely, let θ_k be the angle between $s^{(k)}$ and the negative gradient direction of $\|r\|$ at $x^{(k)}$. Then, according to [98], in the worst case,

$$\frac{1}{\kappa(r'_k)} \leq \cos(\theta_k) \leq \frac{2}{\kappa(r'_k)}, \tag{5.9}$$

where $\kappa(r'_k)$ is the condition number of $r'(x^{(k)})$ and $\cos(\theta_k) = \frac{-(s^{(k)})^T \nabla f(x^{(k)})}{\|s^{(k)}\| \, \|\nabla f(x^{(k)})\|}$. This means that the Newton direction can be nearly orthogonal to the gradient of $\|r\|$ when $\kappa(r'_k)$ is large. In the incompressible Navier-Stokes equations, $\kappa(r'_k)$ becomes very large when Re is high or when the mesh size is fine. A closer look at (5.9) and its proof in [98] shows that the linear preconditioner M_k^{-1} in (5.2) does not appear in the estimate (5.9), which means that even though the linear preconditioning may speed up the solution algorithm for the Jacobian system, it does not help improve the quality of the search direction. Therefore, we believe that to enhance the robustness of INB by finding a better search direction, the preconditioner has to be *nonlinear*. An alternative approach to improving the quality of the search direction is based on the affine invariant Newton methods [32] using the natural monotonicity test for highly nonlinear systems.

Another source of the failure or slow convergence of INB is believed to be the unbalanced nonlinearity of systems of equations. In some cases, the system $r(x)$ can be split into two subsystems $[\hat{r}_1(\hat{x}_1, \hat{x}_2), \hat{r}_2(\hat{x}_1, \hat{x}_2)]^T$ and $x = (\hat{x}_1, \hat{x}_2)^T$, where $\hat{r}_1$ is more nonlinear than $\hat{r}_2$. The systems with unbalanced nonlinearity often arise from the discretizations of PDEs in many physical and engineering applications, for example, shocks or non-smooth fronts in compressible flows, and interior or boundary layers in high Reynolds incompressible flows [19, 20, 21, 22, 104]. The algebraic subsystems associated with the non-smooth regions are expected to be much more difficult to solve than smooth regions. In that case, the damping factor $\lambda^{(k)}$ is usually kept extremely small by the strongest nonlinearity of components except near convergence [104] so that stagnation of INB is often observed. If $\lambda^{(k)}$ is smaller than some tolerances, INB fails. The nonlinear elimination method [74, 104, 103] is an approach to overcoming such problems. In the nonlinear elimination method, we eliminate the troublesome component $\hat{r}_1$ from the whole system $r(x)$. This includes two nonlinear solve processes: First $\hat{r}_1(h(\hat{x}_2), \hat{x}_2) = 0$ is solved for evaluating an implicit function $h(\hat{x}_2)$ for any given $\hat{x}_2$. Then, once $h(x_2)$ is determined, INB is applied for solving $s(\hat{x}_2) \equiv \hat{r}_2(h(\hat{x}_2), \hat{x}_2) = 0$.

Although the efficiency of the nonlinear elimination has been shown for certain problems with proper choices of $\hat{r}_1$ [74, 104, 103], the major drawback of the method is that in practice it is hard to determine how to select $\hat{r}_1$ for elimination without any prior knowledge of the solution behavior. On the other hand, the technique of nonlinear preconditioning, motivated by the nonlinear elimination method, is a more general approach to nonlinear systems with unbalanced nonlinearities. To demonstrate how nonlinear preconditioning improves the robustness of INB, in the next section, we use a two-dimensional nonlinear system with unbalanced nonlinearity as an example. Through nonlinear preconditioning based on the additive Schwarz framework rather than prior knowledge of the solution

behavior, the original nonlinear system is reformulated into a new system with a more balanced nonlinearity. Then INB is applied to solve the preconditioned system.

5.2.4 A simple example with unbalanced nonlinearity

Let $r(x) : R^2 \to R^2$, and consider two-variable nonlinear systems

$$r(x_1, x_2) = \begin{pmatrix} r_1(x_1, x_2) \\ r_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} (x_1 - x_2^3 + 1)^m - x_2^m \\ x_1 + 2x_2 - 3 \end{pmatrix} = 0, \tag{5.10}$$

where $m = 1, 3$, and 5. The nonlinear systems (5.10) have a root at $x = (1, 1)^T$ for any values of m. Obviously, the nonlinearity of these systems are unbalanced, especially for large m; for a given x_1 or x_2, the first equation becomes much more difficult to solve than the second. Using the technique of nonlinear preconditioning, we reformulate the original systems (5.10) to a new preconditioned system as follows.

Let $t(x) : R^2 \to R^2$ be nonlinearly preconditioned function of $r(x)$ given by $[t_1(x_1, x_2), t_2(x_1, x_2)]^T$, where t_1 and t_2 are defined as the solution of $r_1(x_1 - t_1, x_2) = 0$ and $r_2(x_1, x_2 - t_2) = 0$, respectively. Note that this is a special case of Schwarz nonlinear preconditioner, which is introduced in the next section, with zero overlap. In practice, it may be neither possible nor necessary to form $r(x)$ explicitly. But for this simple example, $t(x) = 0$ can be written explicitly by solving t_1 and t_2 in terms of x_1 and x_2, which is

$$t(x) = \begin{pmatrix} t_1(x_1, x_2) \\ t_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} x_1 - x_2 - x_2^3 + 1 \\ x_1 + 2x_2 - 3 \end{pmatrix} = 0 \tag{5.11}$$

Note that the preconditioned system has the same form for any m. It is easy to check that $x = (1, 1)^T$ also is a solution of the preconditioned system (5.11). For

Table 5.1: Number of Newton iterations: The exact Newton method is applied for original systems and preconditioned system, starting from four different initial guesses.

	Original systems (5.10)			**Preconditioned system (5.11)**
$x^{(0)}$	$m = 1$	$m = 3$	$m = 5$	
$(0, 0)^T$	5	10	14	5
$(0, 2)^T$	5	28	1080	5
$(2, 0)^T$	5	1	9	5
$(2, 2)^T$	5	7	10	5

the case of $m = 5$, the second equations in both the original and the new system remain the same, while the powers of the leading terms for x_1 and x_2 in the second equation are reduced considerably. More importantly, the nonlinearity of the two equations in the new system is more balanced than it is in the original system.

The exact Newton method is applied for solving both systems (5.10) and (5.11), starting from four different initial guesses, $x^{(0)} = (0, 0)^T$, $(0, 2)^T$, $(2, 0)^T$, and $(2, 2)^T$. As shown in Table 5.1, we see that the exact Newton method becomes harder to converge as the value of m increases and the convergence of the Newton method for the original systems is sensitive to the choices of an initial guess, especially for $m = 5$. Meanwhile, the preconditioned system requires fewer Newton iterations than original system does and the Newton method with nonlinear preconditioning is very robust so that the choices of a good initial guess becomes less important for the convergence of the Newton method. Figures 5.1 and 5.2 show the contours of the merit functions and the histories of Newton iterations for the original system (5.10) with m=5 and the preconditioned system (5.11). Starting from $x^{(0)} = (0, 2)^T$, INB for the original system has some trouble in the region $[0, 0.5] \times [1.2, 1.6]$, taking almost 1000 iterations.

Figure 5.1: Contour plot of $\log(0.5||r(x)||_2^2 + 1)$ with $m = 5$ and the histories of Newton iterations. In the case of $x^{(0)} = (0, 2)^T$, every 50 Newton iterations is denoted by a mark. In the other cases, there is a mark for each Newton iteration.

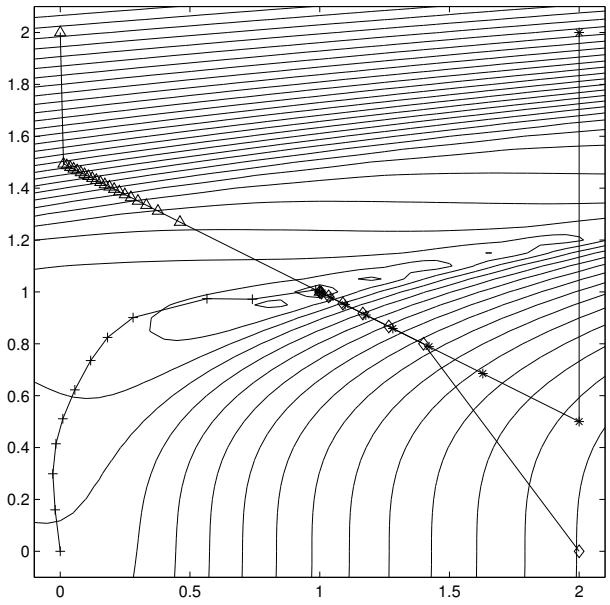
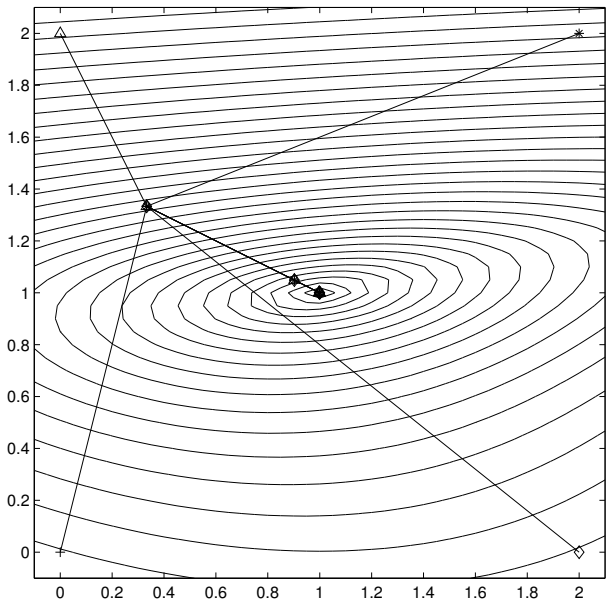


Figure 5.2: Contour plot of $\log(0.5||t(x)||_2^2 + 1)$ and the histories of Newton iterations. In all cases, each Newton iteration is denoted by a mark.



5.3 One-level ASPIN algorithm for incompressible Navier-Stokes equations

When INB is used directly on (3.14), very good results have been observed for the low Reynolds number cases. However, when the Reynolds number is high, INB sometimes converges nicely, but sometimes, it does not converge. The convergence depends on many unknown factors, in other words, the robustness is not guaranteed. In the rest of this section, we re-formulate the nonlinear system (3.14) in the form of preconditioning, and then apply INB to the nonlinearly preconditioned system.

5.3.1 Nonlinear additive Schwarz preconditioning

We introduce the nonlinear preconditioner by defining four key components: the velocity and pressure spaces associated with subdomains; the restriction and interpolation operators; the subdomain nonlinear functions; and the subdomain correction operators. There are different ways to define a nonlinear preconditioner, but here we consider only the additive Schwarz framework. We will show numerically in section 5 that nonlinear additive Schwarz [37] is an excellent nonlinear preconditioner in the sense that it enhances the robustness of INB for a wide range of Reynolds numbers and reduces the number of both linear and nonlinear iterations.

To define parallel Schwarz type preconditioners [96] we need to partition the finite element mesh $\mathcal{T}^h$ introduced in the previous section. Let $\{\Omega_i^h, i = 1, ..., N\}$ be a non-overlapping subdomain partition whose union covers the entire domain Ω and its mesh $\mathcal{T}^h$. We use $\mathcal{T}_i^h$ to denote the collection of mesh points in Ω_i^h. To obtain overlapping subdomains, we expand each subdomain Ω_i^h to a larger subdomain $\Omega_i^{h,\delta}$ with the boundary $\partial\Omega_i^{h,\delta}$. Here δ is an integer indicating the degree

of overlap. We assume that $\partial\Omega_i^{h,\delta}$ does not cut any elements of $\mathcal{T}^h$. Similarly, we use $\mathcal{T}_i^{h,\delta}$ to denote the collection of mesh points in $\Omega_i^{h,\delta}$.

Now, we define the subdomain velocity space as

$$V_i^h = \{v^h \in V^h \cap (H^1(\Omega_i^{h,\delta}))^2 : v^h = 0 \text{ on } \partial\Omega_i^{h,\delta}\}$$

and the subdomain pressure space as

$$P_i^h = \{p^h \in P^h \cap L^2(\Omega_i^{h,\delta}) : p^h = 0 \text{ on } \partial\Omega_i^{h,\delta}\backslash\Gamma\}.$$

On the physical boundaries, we impose Dirichlet conditions according to the original equations (3.1). On the artificial boundaries, we assume both $\mathbf{u} = 0$ and $p = 0$. Similar boundary conditions were used in [71].

Let $R_i : V^h \times P^h \to V_i^h \times P_i^h$ be a restriction operator, which returns all degrees of freedom (both velocity and pressure) associated with the subspace $V_i^h \times P_i^h$. R_i is an $3n_i \times 3n$ matrix with values of either 0 or 1, where n and n_i are the total number of mesh points in $\mathcal{T}^h$ and $\mathcal{T}_i^{h,\delta}$, respectively, and $\sum_{i=1}^N 3n_i \geq 3n$. Note that for $Q_1 - Q_1$ elements, we have three variables per mesh point, two for the velocity and one for the pressure. Then, the interpolation operator R_i^T can be defined as the transpose of R_i. The multiplication of R_i (and R_i^T) with a vector does not involve any arithmetic operation, but does involve communication in a distributed memory parallel implementation.

Using the restriction operator, we define the subdomain nonlinear function $F_i : R^{3n} \to R^{3n_i}$ as

$$F_i = R_i F.$$

We next define the subdomain mapping functions, which in some sense play the role of subdomain preconditioners. For any given $x \in R^{3n}$, $T_i(x) : R^{3n} \to R^{3n_i}$ is defined as the solution of the following subspace nonlinear systems,

$$F_i(x - R_i^T T_i(x)) = 0, \text{ for } = 1, ..., N. \tag{5.12}$$

Throughout this paper, we assume that (5.12) is uniquely solvable. Using the subdomain mapping functions, we introduce a new global nonlinear function,

$$\mathcal{F}^{(1)}(x) = \sum_{i=1}^{N} R_i^T T_i(x), \tag{5.13}$$

which we refer to as the nonlinearly preconditioned $F(x)$. The one-level additive Schwarz inexact preconditioned Newton algorithm (ASPIN(1)) is defined as: Find the solution x^* of (3.14) by solving the nonlinearly preconditioned system,

$$\mathcal{F}^{(1)}(x) = 0, \tag{5.14}$$

using INB with an initial guess $x^{(0)}$.

For strong elliptic problems, it can be shown that nonlinearly preconditioned system and the original system have the same solution [20]. For the Navier-stokes equations, we will verify numerically, in subsection 5.4.3, that the preconditioned nonlinear system,

$$\mathcal{F}^{(1)}(x) = 0 \tag{5.15}$$

has the same solution as that of the original system (3.14).

5.3.2 Computing the Jacobian of the preconditioned system

Several techniques are available for the construction and solving the Jacobian problems. Some methods are matrix-free, and some need the explicit construction of the matrix. In our implementation, we choose to construct the Jacobian of $\mathcal{F}^{(1)}(x)$ semi-explicitly. Since $\mathcal{F}^{(1)}(x)$ is defined implicitly in (5.12) and (5.13), the Jacobian calculation is not as straightforward as that of $F(x)$. Here, we describe a computable form of the Jacobian matrix for the preconditioned system as suggested in [20].

Consider subdomain $\Omega_i^{h,\delta}$. Let $\left(\Omega_i^{h,\delta}\right)^c$ be the complement of $\Omega_i^{h,\delta}$ in the domain Ω. We write $x = (x_i, x_i^c)$, where $x_i = R_i x$ and $x_i^c = R_i^c x$. Here R_i^c is a

restriction matrix defined similarly as R_i. From the definition (5.12) of $T_i(x)$, we have

$$F_i(x_i - T_i(x_i, x_i^c), x_i^c) = 0. \tag{5.16}$$

Let $y_i = x_i - T_i$ and $z_i = (y_i, x_i^c)$, then the global function can be rewritten as a function of z_i:

$$F(z_i) = F(y_i, x_i^c).$$

Furthermore, we write the Jacobian matrix of the original nonlinear function $F(x)$ in the form of

$$J = \left(\frac{\partial F}{\partial z_i}\right)_{n\times n}$$

or, in terms of y_i and x_i^c as

$$J = \left(\frac{\partial F}{\partial y_i}\right) R_i + \left(\frac{\partial F}{\partial x_i^c}\right) R_i^c,$$

since subdomains $\Omega_i^{h,\delta}$ and $\left(\Omega_i^{h,\delta}\right)^c$ do not overlap.

Similarly, let $J_i(z_i)$ be the Jacobian of the nonlinear function $F(\cdot)$ restricted to the subdomain $\Omega_i^{h,\delta}$, i.e.,

$$J_i(z_i) = \left(\frac{\partial F_i}{\partial y_i}\right)_{n_i\times n_i}.$$

Suppose that we want to evaluate the Jacobian of the preconditioned function, denoted as $\mathcal{J}$, at the kth Newton iteration, $x^{(k)} = (x_i^{(k)}, (x_i^c)^{(k)})$. Using (5.13), the Jacobian $\mathcal{J}$ can be calculated as follows:

$$\mathcal{J}^{(1)} \equiv (\mathcal{F}^{(1)})' = \sum_{i=1}^{N} R_i^T \left(\frac{\partial T_i}{\partial x}\right).$$

We next derive an approximate formula for each $\partial T_i/\partial x$. Now, taking the partial derivative of the equation (5.16) with respect to x_i, we find that

$$\left(\frac{\partial F_i}{\partial y_i}\right)\left(\frac{\partial y_i}{\partial x_i}\right) = 0,$$

or, more precisely,

$$\left(\frac{\partial F_i}{\partial y_i}\right)\left(I_i - \frac{\partial T_i}{\partial x_i}\right) = 0.$$

Assuming $\left(\frac{\partial F_i}{\partial y_i}\right)$ is nonsingular, we obtain that

$$\left(\frac{\partial T_i}{\partial x_i}\right) = I_i = \left(\frac{\partial F_i}{\partial y_i}\right)^{-1}\left(\frac{\partial F_i}{\partial y_i}\right). \tag{5.17}$$

Here, I_i is the $n_i \times n_i$ identity matrix. Next, we take the partial derivative of (5.16) with respect to x_i^c to obtain

$$-\left(\frac{\partial F_i}{\partial y_i}\right)\left(\frac{\partial T_i}{\partial x_i^c}\right) + \left(\frac{\partial F_i}{\partial x_i^c}\right) = 0.$$

Solving the above equation for $\left(\frac{\partial T_i}{\partial x_i^c}\right)$ yields

$$\left(\frac{\partial T_i}{\partial x_i^c}\right) = \left(\frac{\partial F_i}{\partial y_i}\right)^{-1}\left(\frac{\partial F_i}{\partial x_i^c}\right). \tag{5.18}$$

Note that

$$\left(\frac{\partial T_i}{\partial x}\right)_{n_i \times n} = \left(\frac{\partial T_i}{\partial x_i}\right)R_i + \left(\frac{\partial T_i}{\partial x_i^c}\right)R_i^c. \tag{5.19}$$

By substituting (5.17) and (5.18) into (5.19), we have

$$\begin{aligned}\left(\frac{\partial T_i}{\partial x}\right) &= \left(\frac{\partial F_i}{\partial y_i}\right)^{-1}\left[\left(\frac{\partial F_i}{\partial y_i}\right)R_i + \left(\frac{\partial F_i}{\partial x_i^c}\right)R_i^c\right] \\ &= \left(\frac{\partial F_i}{\partial y_i}\right)^{-1} R_i\left[\left(\frac{\partial F}{\partial y_i}\right)R_i + \left(\frac{\partial F}{\partial x_i^c}\right)R_i^c\right] \\ &= (J_i)^{-1}R_i J. \end{aligned} \tag{5.20}$$

Summing up (5.20) for all subdomains and evaluating x at $x^{(k)}$, we have a formula for the Jacobian of the preconditioned nonlinear function

$$\mathcal{J}^{(1)} = \sum_{i=1}^{N}\left[R_i^T(J_i(z_i))^{-1}R_i\right]J(z_i)\Big|_{z_i=(x_i^{(k)}-T_i(x^{(k)}),(x_i^c)^{(k)})}. \tag{5.21}$$

Although (5.21) is computable, in practice, it is more convenient to use the following approximation suggested in [20]:

$$\widehat{\mathcal{J}}^{(1)} = \sum_{i=1}^{N} \left[R_i^T (J_i(z))^{-1} R_i \right] J(z)\Big|_{z=x^{(k)}} \tag{5.22}$$

Remark 4 Note that the subdomain preconditioner $(J_i)^{-1}$ corresponds to the solution of a discrete Stokes-like problem using GLS formulation with homogenous Dirichlet boundary conditions for both the velocity and pressure. In our implementation, J_i is obtained from $J_i = R_i J R_i^T$, where J is constructed using a multi-colored forward finite difference method.

Remark 5 Although each component of $\widehat{\mathcal{J}}^{(1)}$ is sparse, $\widehat{\mathcal{J}}^{(1)}$ itself is often dense and expensive to form explicitly. However, if a Krylov subspace method is used to the Jacobian problem, only the Jacobian-vector product, $u = \widehat{\mathcal{J}}^{(1)} v$, is required. In a distributed-memory parallel implementation, this operation consists of four phrases:

(1) Perform the matrix-vector multiply, $w = Jv$, in parallel.

(2) On each subdomain, collect the data from the subdomain and its neighboring subdomains, $w_i = R_i w$.

(3) Solve $J_i u_i = w_i$ using a sparse direct solver.

(4) Send the partial solutions and its neighboring subdomain and take the sum, $u = \sum_{i=1}^{N} R_i^T u_i$.

5.3.3 Details of one-level additive Schwarz preconditioned inexact Newton

Summarizing the discussions in subsections 5.3.1 and 5.3.2, we here describe the complete ASPIN(1) algorithm for solving incompressible Navier-Stokes equations. Let $x^{(0)}$ an initial guess and $x^{(k)}$ be the current approximate solution. Then

a new approximate solution $x^{(k+1)}$ can be computed by the ASPIN(1) algorithm as follows:

Algorithm 2 (ASPIN(1)) .

Step 1: Evaluate the nonlinear residual $\mathcal{F}^{(1)}(x)$ at $x^{(k)}$ through the following steps:

(1) Find $w_i^{(k)} = T_i(x^{(k)})$ by solving, in parallel, the local subdomain nonlinear systems

$$G_i(w) \equiv F_i(x^{(k)} - R_i^T w) = 0,$$

using INB with the initial guess $w = 0$.

(2) Form the global residual

$$\mathcal{F}^{(1)}(x^{(k)}) = \sum_{i=1}^{N} R_i^T w_i^{(k)}.$$

Step 2: Check the stopping condition on $||\mathcal{F}^{(1)}(x^{(k)})||_2$. If $||\mathcal{F}^{(1)}(x^{(k)})||_2$ is small enough, stop, otherwise, continue.

Step 3: Evaluate pieces of the Jacobian matrix $\widehat{\mathcal{J}}^{(1)}(x)$ of the preconditioned system that are needed in order to multiply (5.23) below with a vector in the next step. This includes $J(x^{(k)})$ as well as $J_{\Omega_i'}$ and its sparse LU factorization.

$$\widehat{\mathcal{J}}^{(1)} = \sum_{i=1}^{N} \left[R_i^T (J_i(x^{(k)}))^{-1} R_i \right] J(x^{(k)}). \tag{5.23}$$

Step 4: Find an inexact Newton direction $s^{(k)}$ by solving the following Jacobian system approximately using a Krylov subspace method

$$\widehat{\mathcal{J}}^{(1)} s^{(k)} = -\mathcal{F}^{(1)}(x^{(k)})$$

in the sense that

$$||\mathcal{F}^{(1)}(x^{(k)}) + \widehat{\mathcal{J}}^{(1)}(x^{(k)})s^{(k)}||_2 \leq \eta_k ||\mathcal{F}^{(1)}(x^{(k)})||_2$$

for some $\eta_k \in [0, \eta_{max}]$ for some $\eta_{max} < 1$ independent of k.

Step 5: Scale the search direction $s^{(k)} \leftarrow \frac{s_{max}}{||s^{(k)}||_2} s^{(k)}$ if $||s^{(k)}||_2 \geq s_{max}$.

Step 6: Compute a new approximate solution

$$x^{(k+1)} = x^{(k)} + \lambda^{(k)} s^{(k)},$$

where $\lambda^{(k)}$ is a damping parameter determined by the standard backtracking procedure described in (5.5).

Remark 6 At the kth global nonlinear iteration, local subdomain nonlinear systems

$$G_i(w) = 0, i = 1, \ldots, N$$

need to be solved. We solve these subsystems using INB. During local nonlinear iterations, a direct sparse solver, LU decomposition, is employed for solving each local Jacobian system.

Remark 7 No preconditioning is used in Step 3 of Algorithm 2. In fact, $\widehat{\mathcal{J}}^{(1)}$ can be viewed as the original Jacobian system J preconditioned by a one-level additive Schwarz preconditioner. Hence, $\widehat{\mathcal{J}}^{(1)}$ is well-conditioned through nonlinear preconditioning as long as the number of subdomains is not very large.

Remark 8 As suggested by Dennis and Schnabel in the page 129 of [29], we include the re-scaling of the search direction $s^{(k)}$ in Step 4 if $||s^{(k)}||_2 \geq s_{max}$. The purposes of this step length constraint are to avoid very large steps during calculation and to prevent the intermediate solution from leaving the domain of

our interest. The scalar s_{max} is provided by the user. In the subsection , we will see that the re-scaling step plays an important role in enhancing the robustness of ASPIN(1) for solving incompressible Navier-Stokes equations, especially when Re is high. With careful choices of s_{max}, the efficiency of ASPIN(1) can be improved as well.

Remark 9 One key issue of the backtracking technique is the selection of the merit function. For the global nonlinear problem, we use $f(x) = ||\mathcal{F}^{(1)}(x)||_2^2/2$; for the local nonlinear problem, we use $f(x) = ||F_i(x)||_2^2/2$.

5.4 Numerical Results: I. Comparison of one-level ASPIN and NKS

In this section, we present a few numerical results using ASPIN(1) to show its convergence properties, robustness with respect to high Reynolds numbers, and parallel scalability. We also compare the results with those obtained using a standard Newton-Krylov-Schwarz algorithm [19, 73]. A driven cavity flow problem [49] and a backward-facing step problem [47] are considered here as benchmarks to evaluate the performance of the algorithms. The implementation uses PETSc [6], and all numerical results are obtained on a cluster of distributed-memory workstations. Double precision is used throughout the computation. A zero initial guess is used for all test cases. Only machine-independent results are reported. Since current version of our software has not fully optimized, only some preliminary timing results obtained by two-level ASPIN is reported in next section.

5.4.1 Selection of parameters for one-level ASPIN

ASPIN(1) is a collection of several nested linear and nonlinear solvers and many stopping parameters are involved. We summarize the parameters selected in our numerical experiments as follows:

- The global nonlinear iteration is stopped if the condition,

$$||\mathcal{F}^{(1)}(x^{(k)})||_2 \leq \varepsilon_{global-nonlinear}||\mathcal{F}^{(1)}(x^{(0)})||_2,$$

is satisfied. We set $\varepsilon_{global-nonlinear} = 10^{-6}$ for all test cases, and the success of ASPIN(1) is declared when the above condition is satisfied.

- GMRES is used for solving the global Jacobian systems. The global linear iteration is stopped if the condition,

$$||\mathcal{F}^{(1)}(x^{(k)}) + \widehat{\mathcal{J}}^{(1)}(x^{(k)})S^{(k)}||_2 \leq \varepsilon_{global-linear}||\mathcal{F}^{(1)}(x^{(k)})||_2,$$

is satisfied. We vary $\varepsilon_{global-linear}$ ranging from 10^{-3} to 10^{-6} and find that the global Jacobian systems need to be solved with a certain degree of accuracy, e.g. no larger than 10^{-4}, to guarantee the robustness and efficiency of ASPIN(1) for some cases. We also observe that less than 8% of linear iterations can be saved for most cases using larger acceptable $\varepsilon_{global-linear}$. Therefore, to assure that ASPIN(1) converges over a wide range of our applications, we select $\varepsilon_{global-linear} = 10^{-6}$ for all test cases.

- The local nonlinear iteration on each subdomain is stopped if the condition,

$$||G_i(W_{i,l}^{(k)})||_2 \leq \varepsilon_{local-nonlinear}||G_i(W_{i,0}^{(k)})||_2,$$

is satisfied, or if the maximum local nonlinear iteration of 25 is reached. We test different $\varepsilon_{local-nonlinear}$ ranging from 10^{-4} to 10^{-6} and find that the numbers of ASPIN iterations are nearly independent of $\varepsilon_{local-nonlinear}$, but the number of local nonlinear iterations per global function evaluation, which is one of the expensive steps in the algorithm, can be reduced by choosing a loose tolerance. Hence, $\varepsilon_{local-nonlinear} = 10^{-4}$ is set for all test cases.

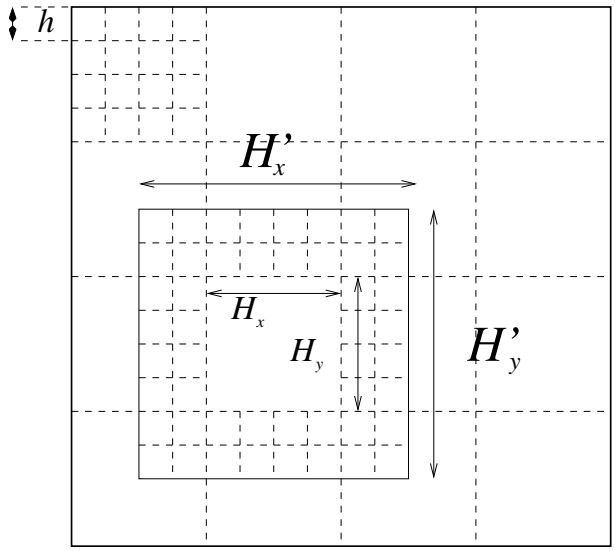


Figure 5.3: A sample mesh partition with $ovlp = 2$ on a rectangular mesh.

- Regular checkerboard partitions are used for our experiments. The number of subdomains is always the same as the number of processors, n_p.

- The overlapping size is defined as

$$ovlp = \max\left\{\frac{H'_x - H_x}{2h}, \frac{H'_y - H_y}{2h}\right\}$$

for both interior subdomains and subdomains touching the boundary. Rectangular elements are used for the two benchmark problems. H'_x and H'_y are defined here as the side lengths of the overlapping subdomain $\Omega_i^{h,\delta}$ in x-direction and y-direction, respectively. Similarly, H_x and H_y are defined as the side lengths of the non-overlapping subdomain Ω_i^h in x-direction and y-direction, respectively. A graphic example is presented in Figure 5.3. We use $ovlp = 2$ for all test cases, except in Table 5.10, where we study the effect of using different overlapping size.

- The local subdomain Jacobian matrix G'_i, and the global Jacobian matrix J are formed using multi-colored forward finite differences. The finite

difference parameter is set at 10^{-8}.

- We use the standard backtracking technique for both global and local nonlinear problems. The parameters associated with INB are: $\alpha = 10^{-4}$, $\lambda_{\min} = 0.1$, and $\lambda_{\max} = 0.5$.

5.4.2 A Newton-Krylov-Schwarz algorithm

We compare our algorithm with an inexact Newton based algorithm applied directly to the original nonlinear system (3.14). There are several parallel Newton-Krylov algorithms including Newton-Krylov-Multigrid [38, 85] and Newton-Krylov-Schwarz(NKS) [19, 73, 94]. Because of the similar data structure, we compare only with an NKS algorithm. NKS has three main components: a Newton method as the nonlinear solver, a Krylov subspace method as the linear solver, and a Schwarz-type method as the preconditioner. For the nonlinear solver, the INB described in Section 3 is employed. Three choices of forcing term η_k are tested. Choices 1 and 2 are given by (5.3) and (5.4). We denote the constant $\eta_k = 10^{-6}$ as Choice 0. For the linear solver, we apply GMRES for solving each Jacobian system. To accelerate the convergence of GMRES, we choose a one-level additive Schwarz method as a right preconditioner:
At each Newton iteration, find a Newton direction $S^{(k)}$ to satisfy

$$||F(x^{(k)}) + (J(x^{(k)})M^{-1})(MS^{(k)})||_2 \leq \eta_k ||F(x^{(k)})||_2, \tag{5.24}$$

where $M^{-1} = \sum_{i=1}^{N} R_i^T J_i^{-1}(x^{(k)}) R_i$. Note that right preconditioning preserves the l_2 norm so that the preconditioned system (5.24) changes neither the linear residual norm nor the function norm. We use the same partition and overlap as in the corresponding ASPIN(2) algorithm. The list of parameters for Choice 1 and Choice 2 appears in Table 5.2 [41, 42]. We declare the convergence of INB if the

Table 5.2: Parameters for Choice 1 and Choice 2

For both Choice 1 and Choice 2	
Initial forcing term η_0	0.01
Maximum forcing term η_{max}	0.9
For Choice 2 only	
Power ρ	2.0
Multiplication factor γ	0.9

condition,

$$||F(x^{(k)})||_2 \leq \varepsilon_{nonlinear}||F(x^{(0)})||_2,$$

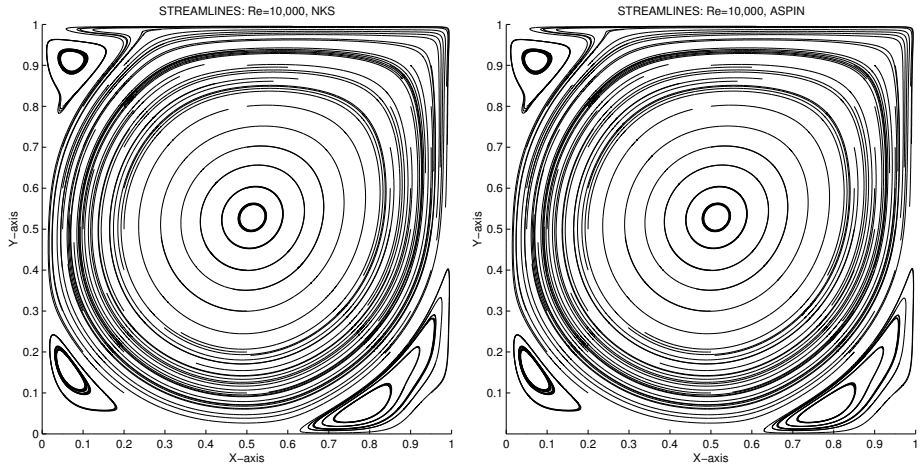
is satisfied. Here $\varepsilon_{nonlinear} = 10^{-6}$ for all test cases. Otherwise, we claim that NKS fails. This happens if the maximum nonlinear iteration of 100 is reached, or the backtracking fails.

5.4.3 Test 1: Lid driven cavity flow

We consider the incompressible lid driven cavity flow on the unit square described in Chapter 3. We run the test for three uniform meshes 64×64, 128×128, and 256×256. The subdomains are obtained by partitioning the mesh uniformly as shown in Figure 5.3. For this test case, we consider 2×2 and 4×4 subdomain partitions. As mentioned before, the number of processors is the same as the number of subdomains.

Numerical verification of the computed solutions We claim that two systems of equations are equivalent if they have the same solution. To show that the original system and the preconditioned system are equivalent is trivial for the case of linear preconditioning, but not so for the case of nonlinear preconditioning. In [20], Cai and Keyes proved analytically the equivalence of two nonlinear systems, the original system and nonlinearly preconditioned system, under cer-

Figure 5.4: Streamlines. The original (left) and preconditioned (right) nonlinear system.



tain assumptions. However, applying the equivalence theorem in [20] for general nonlinear systems $F(x) = 0$ such as (3.14) is not straightforward since it is very difficult to check whether the assumptions hold or not for our cases. Instead, we verify numerically that the original nonlinear system (3.14) and the preconditioned system (5.15) have the same solution by comparing two sets of solution plots. Figures 5.4 and 5.5 are two typical sets of solution plots. Clearly, the streamlines in Figure 5.4 and the pressure elevations in Figure 5.5 obtained from solving two systems for $Re = 10,000$ on a 256×256 mesh are almost indistinguishable. Hence, nonlinear preconditioning does not alter the solution of the nonlinear system; we believe that the minor differences are due to the use of preconditioner-dependent stopping conditions. Note that the original nonlinear system is solved by the NKS algorithm with a zero-order Reynolds number based continuation technique [53], since NKS itself fails to converge for such a high Reynolds number.

Choices of s_{max} In general, the range of feasible values for s_{max} is dependent on some physical parameters, such as the Reynolds number, and other

Figure 5.5: Pressure elevations. The original (top) and preconditioned (bottom) nonlinear system.

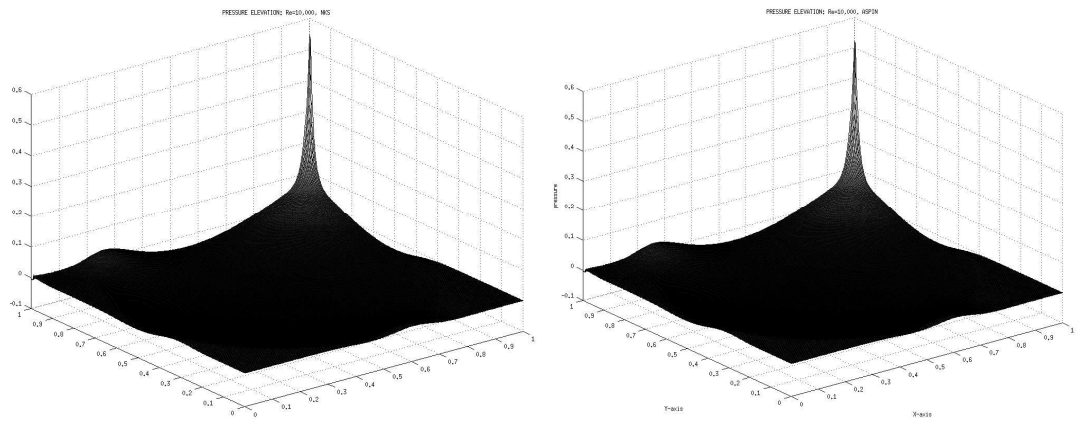


Table 5.3: Feasible values for s_{max} and optimal values for s_{max}. A 128×128 mesh on 16 processors for $Re = 1,000, 5,000$ and $10,000$. The intervals of feasible values for S_{max} are multiplied by the first step length $||s^{(1)}||_2$.

Re	$\|\|s^{(1)}\|\|_2$	Feasible values for S_{max}	Optimal value for S_{max}
10^3	694	[0.003, 0.865]	300–450
5×10^3	156	[0.013, 0.160]	25
10^4	57	[0.018, 0.044]	2.5

parameters, such as the mesh size and the subdomain size. The optimal value for s_{max} is determined empirically. "Optimal" refers to the idea that s_{max} is selected so that ASPIN(1) requires the minimum number of global nonlinear iterations in the several successful runs. Figure 5.6 shows the history of nonlinear residuals of ASPIN(1) with different s_{max} for $Re = 10,000$. Note that ASPIN(1) fails to converge beyond $s_{max} = 3$ in this case. From the plot, we see that although small S_{max} enhances the robustness of ASPIN(1), it slows down the overall convergence. The number of ASPIN(1) iterations is decreased as we increase s_{max}. Up to 50% nonlinear iterations can be saved if we choose the optimal value for s_{max}. Also, as shown in Table 5.3, the range of feasible s_{max} is much wider for a low Reynolds number flow than for a high Reynolds number flow. In other words, for high Reynolds number flows, choosing a feasible value of s_{max} to assure the convergence of ASPIN(1) is more difficult. In that case, it is safer to select small enough s_{max}, compared to the first step length $||s^{(1)}||_2$, to assure the convergence of ASPIN(1) at the first trial, and then the performance of ASPIN(1) can be improved by increasing s_{max} gradually. Note that all numerical results for ASPIN(1) presented later in this section are obtained by using the optimal values for s_{max}.

Comparison with NKS

In Figures 5.7 and 5.8, we compare the nonlinear residual history of AS-

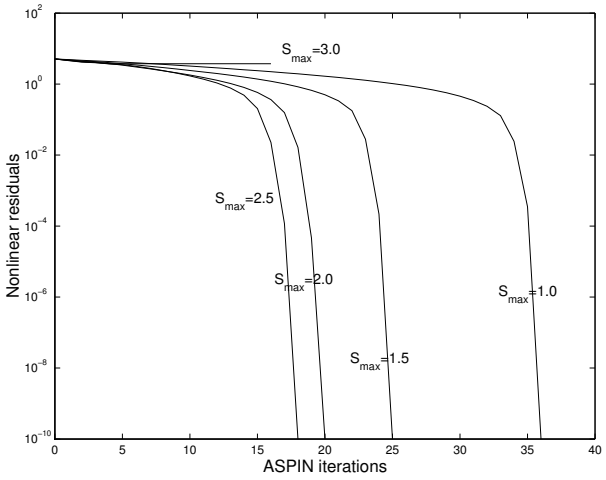


Figure 5.6: History of nonlinear residuals of ASPIN(1) with different values of s_{max}. A 128×128 mesh is used on 16 processors, $Re = 10,000$. ASPIN(1) with $s_{max} = 3.0$ is terminated at the earlier iteration because of the failure of backtracking.

PIN(1) with those of NKS with three different choices of forcing terms. We run ten tests for Reynolds numbers ranging from 10^3 to 10^4, with an increment of 10^3. All results are obtained by using a 128×128 mesh on 16 (4×4) processors. We see that nonlinear residuals of NKS with all choices of forcing terms behave similarly. Except for a few cases with low Reynolds number, NKS nonlinear residuals stagnate around 10^{-3} without any progress after about the first 15 iterations. All of them fail to converge after 100 iterations. Different choices of forcing terms do not help much in this particular set of tests. We should note that the success of NKS using these two adaptive forcing terms on the driven cavity problem up to $Re = 10^4$ has been reported by [94]. Several parameters in NKS need to be well-tuned in order for the method to converge for all applications. By comparing our implementation with [94], we find that the differences in the selections of some algorithmic parameters, such as quality of subdomain solve (exact or inexact), the number of subdomain, and the degree of overlapping, and the choices of some discretization parameters, such as stabilization parameter, τ, may affect the performance of NKS. As we will see in next test case, NKS with adaptive forcing terms seems sensitive to the change in the number of processors (or subdomains). It may be interesting to study how other factors affect the performance of NKS. On the other hand, ASPIN(1) converges for the whole range of Reynolds numbers. Furthermore, we find that ASPIN(1) preserves the local quadratic convergence of Newton when the intermediate solution is near the desired solution.

In addition, to understand the robustness of ASPIN(1) and NKS, we next compare the minimum values of $\cos(\theta_k)$ for ASPIN and NKS with different forcing terms in Table 5.4. All results are obtained by on a 64 × 64 and a 128×128 meshes using 16(=4×4) processors. Recall that θ_k is the angle between the Newton direction $s^{(k)}$ and the negative gradient direction of $\|F(x)\|_2$ at $x^{(k)}$. The values marked with asterisks in the table indicate that NKS fails to converge either after

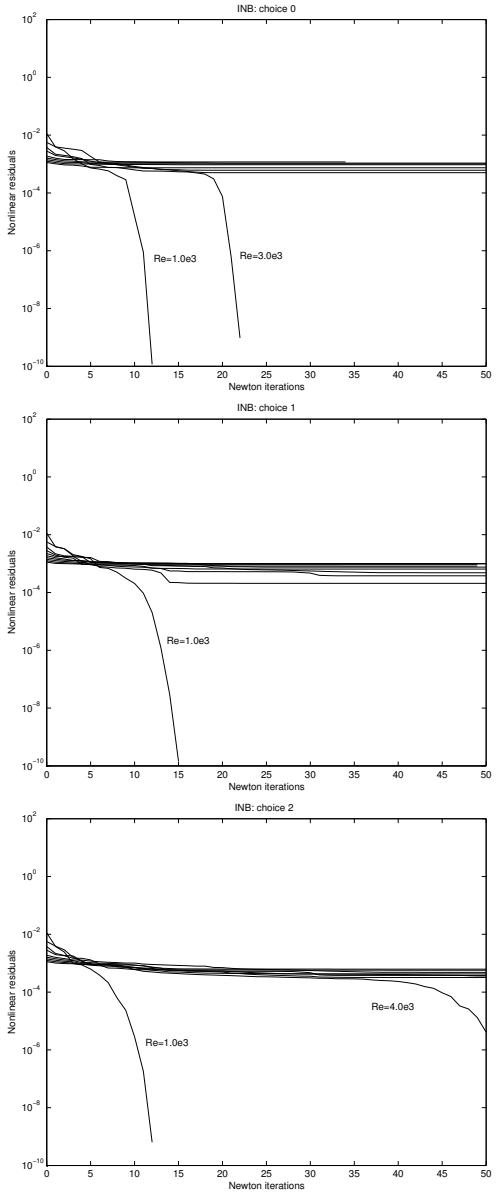


Figure 5.7: History of nonlinear residuals. NKS with three different forcing terms. Only converged cases are labelled.

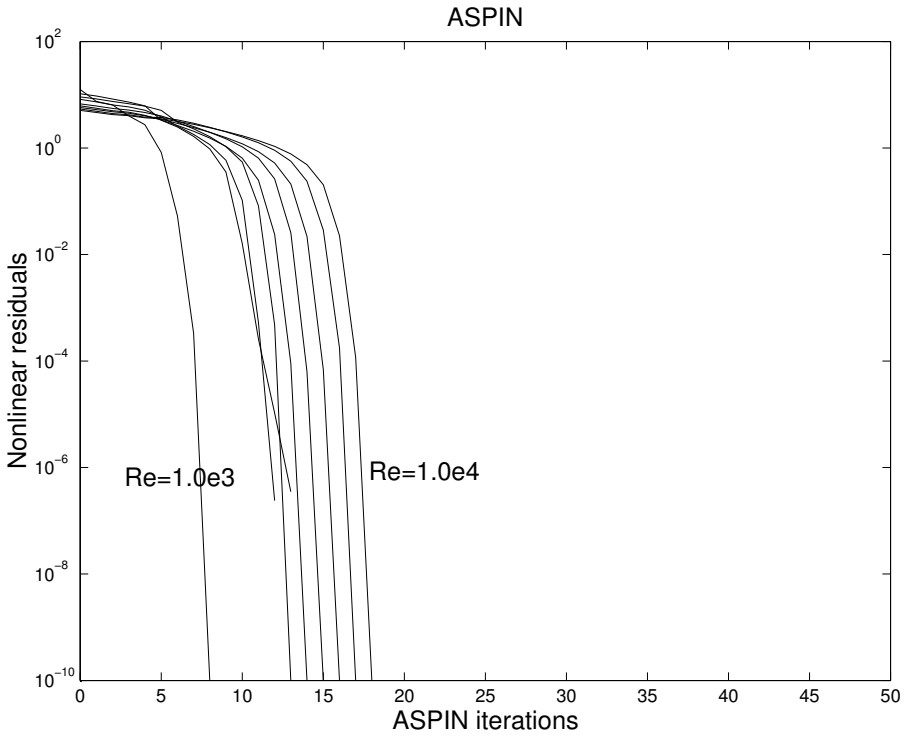


Figure 5.8: History of nonlinear residuals. ASPIN(1) converges in all ten test cases.

Table 5.4: Comparison of the minimum values of $\cos(\theta_k)$ for ASPIN(1) and NKS.

	$Re = 10^3$	$Re = 5 \times 10^3$	$Re = 10^4$
Mesh size: 64×64			
Choice 0	1.68e-03	8.50e-12*	6.70e-11*
Choice 1	4.21e-03	6.22e-08*	1.09e-04*
Choice 2	4.80e-03	4.91e-05*	1.54e-04
ASPIN(1)	7.37e-03	1.74e-03	1.82e-03
Mesh size: 128×128			
Choice 0	8.65e-04	1.97e-07*	3.31e-07*
Choice 1	3.78e-03	3.30e-05*	1.82e-08*
Choice 2	3.33e-03	1.20e-04*	9.27e-05*
ASPIN(1)	2.98e-03	2.94e-03	3.90e-03

100 nonlinear iterations, or the backtracking step fails. For NKS, the minimum value of $\cos(\theta_k)$ is tiny when NKS fails. This agrees well with estimate (5.9), since $\kappa(J(x^{(k)}))$ is expected to be very large for this high Re. On the other hand, the minimum value of $\cos(\theta_k)$ for ASPIN(1) is always away from zero and is not sensitive to the change of Re as well as the refinement of the mesh.

Scalability of ASPIN(1) Scalability is an important issue in parallel computing, and the issue becomes even more significant when we solve large scale problems with many processors. Table 5.5 shows that ASPIN(1) iterations are nearly independent of mesh size; the nonlinear iteration numbers change up or down by small fractions when we increase the mesh size from a coarse mesh, 64×64, to a fine mesh, 256×256 on 16 (4×4) processors. However, the average number of GMRES iterations increases quite a bit when we increase the mesh size. Next we fix the mesh size and vary the number of processors. In Table 5.6, we see that the number of ASPIN(1) iterations does not change much, while the average number of GMRES iterations increases a lot when we increase the number of processors from 4 to 16 on a fixed 128×128 mesh. The increase in GMRES

Table 5.5: Driven cavity problem: Different mesh sizes on 16 processors.

	Number of ASPIN(1) iterations				
mesh sizes	$Re = 10^3$	$Re = 3 \times 10^3$	$Re = 5 \times 10^3$	$Re = 8 \times 10^3$	$Re = 10^4$
64×64	11	12	15	17	18
128×128	14	13	13	16	20
256×256	12	13	18	21	15
	Average number of GMRES iterations				
mesh sizes	$Re = 10^3$	$Re = 3 \times 10^3$	$Re = 5 \times 10^3$	$Re = 8 \times 10^3$	$Re = 10^4$
64×64	86	88	92	97	97
128×128	128	128	132	137	140
256×256	190	188	194	194	197

iteration numbers is not unexpected, since we do not have a coarse space in the preconditioner.

Solving the subdomain nonlinear problems Figure 5.9 shows the numbers of subdomain nonlinear iterations required for different subdomains in Step 1 of Algorithm 2. In this test case, we partition the domain into 2×2 subdomains and number them, first from bottom to top, and then from left to right. Note that two subdomains, Ω_2 and Ω_4, touch the moving lid. We observe that at the beginning of ASPIN(1) iterations, ASPIN(1) has some difficulty solving the subdomain nonlinear problem by INB with a zero initial guess on the subdomain Ω_4, where a boundary layer and singularity present for $Re = 10,000$. INB starts to stall after about 15 iterations and the nonlinear residual cannot be reduced to the order of 10^{-4} on Ω_4 because the size of the subdomain problem is too large. We may resolve this situation by increasing the number of subdomains; there are no unsuccessful subdomain nonlinear iterations observed as we use 4×4 subdomains to solve the same problem. After 7 ASPIN(1) iterations, the nonlinearities of each subdomain become more balanced, and only about up to 4 Newton iterations are

Table 5.6: Driven cavity problem: Different number of processors on a 128×128 mesh.

	Number of ASPIN(1) iterations				
n_p	$Re = 10^3$	$Re = 3 \times 10^3$	$Re = 5 \times 10^3$	$Re = 8 \times 10^3$	$Re = 10^4$
$2 \times 2 = 4$	11	10	13	19	19
$4 \times 4 = 16$	14	13	13	16	20
	Average number of GMRES iterations				
n_p	$Re = 10^3$	$Re = 3 \times 10^3$	$Re = 5 \times 10^3$	$Re = 8 \times 10^3$	$Re = 10^4$
$2 \times 2 = 4$	67	69	71	73	74
$4 \times 4 = 16$	128	128	132	137	140

needed to solve each subdomain problem. ASPIN(1) converges in 19 iterations.

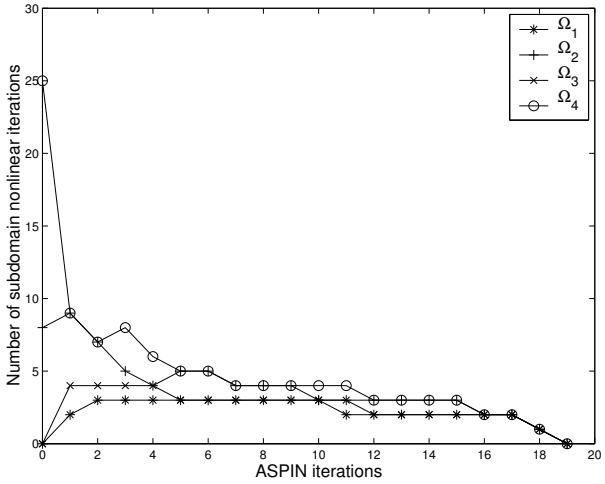


Figure 5.9: Numbers of subdomain nonlinear iterations are required for the global nonlinear function evaluations, which are needed in solving global Jacobian systems on different subdomains. A 128 $\times$128 mesh is tested on 4 processors, $Re = 10,000$.

5.4.4 Test 2: Flow passing a backward-facing step

We consider another benchmark problem, the backward-facing step flow problem described in Chapter 3. We apply two uniform meshes (600×20 and 1200×40). Two subdomain partitions are considered: 10 (5×2) and 20 (5×4) processors. The subdomain ordering for the 20-processor case is shown in Figure 5.10.

Scalability of NKS and ASPIN(1) First, we look at the performance of NKS with three choices of forcing terms when we change the number of processors and the mesh size. In Table 5.7, unlike the driven cavity flow problem, we observe that some choices of forcing term do enhance the robustness of NKS in some cases. For example, NKS with Choice 2 converges for all tested Reynolds numbers on 10 (5×2) and 20 (5×4) processors, while NKS with Choice 0 fails to converge when the Reynolds number is higher than 500. Furthermore, the average GMRES iterations for NKS with Choice 2 is quite small compared with Choice 0 when the convergence can be achieved. However, NKS seems quite sensitive to the changes in the number of processors: The number of Newton iterations for Choice 2 nearly doubles when we increase the number of processors from 10 to 20 in the cases of high Reynolds number. Meanwhile, the number of successes for Choice 1 is reduced from 5 to 2 when we use a stronger preconditioner for the Jacobian system. In Table 5.8, we see that the refinement of the mesh does not increase either the number of Newton iterations or the average GMRES iterations significantly. For high Reynolds numbers, such as $Re = 700$ and 800, NKS with Choice 1 and 2 on a fine mesh requires even fewer numbers of nonlinear iterations than on a coarse mesh.

Next, we study the scalability of ASPIN(1) for the backward-facing step

Table 5.7: Backward-facing step problem: Comparison of NKS with three choices of forcing terms. The number of Newton iterations and the average number of GMRES iterations for different values of Reynolds number. 1200×40 elements on 5×2 and 5×4 processors. "–" indicates a failure of convergence.

	Number of nonlinear iterations					
Choice number	0		1		2	
n_p	5×2	5×4	5×2	5×4	5×2	5×4
$Re = 1 \times 10^2$	6	6	7	7	6	6
$Re = 5 \times 10^2$	–	–	–	20	14	28
$Re = 6 \times 10^2$	–	–	17	29	19	37
$Re = 7 \times 10^2$	–	–	–	44	22	50
$Re = 8 \times 10^2$	–	–	–	43	27	57
	Average number of GMRES iterations					
$Re = 1 \times 10^2$	62	119	20	44	30	48
$Re = 5 \times 10^2$	–	–	–	23	14	25
$Re = 6 \times 10^2$	–	–	13	29	12	24
$Re = 7 \times 10^2$	–	–	–	35	14	25
$Re = 8 \times 10^2$	–	–	–	31	19	28

Table 5.8: Backward-facing step problem: Comparison of NKS with three choices of forcing terms. The number of Newton iterations and the average number of GMRES iterations for different values of Reynolds number. 600×20 and 1200×40 elements on 20 (5×4) processors. "–" indicates a failure of convergence.

	Number of nonlinear iterations					
Choice number	0		1		2	
mesh size	600×20	1200×40	600×20	1200×40	600×20	1200×40
$Re = 1{\times}10^2$	6	6	7	7	6	6
$Re = 5{\times}10^2$	–	–	26	20	22	28
$Re = 6{\times}10^2$	–	–	35	29	33	37
$Re = 7{\times}10^2$	–	–	61	44	63	50
$Re = 8{\times}10^2$	–	–	57	43	72	57
	Average number of GMRES iterations					
$Re = 1{\times}10^2$	62	119	41	44	43	48
$Re = 5{\times}10^2$	–	–	22	23	17	25
$Re = 6{\times}10^2$	–	–	28	29	16	24
$Re = 7{\times}10^2$	–	–	23	35	20	25
$Re = 8{\times}10^2$	–	–	23	31	20	28

Table 5.9: Backward-facing step problem: Different mesh sizes on 20 (5×4) processors.

	Number of ASPIN iterations				
mesh sizes	$Re = 10^2$	$Re = 5\times10^2$	$Re = 6\times10^2$	$Re = 7\times10^2$	$Re = 8\times10^2$
600×20	5	9	15	21	18
1200×40	5	18	19	28	39
	Average number of GMRES iterations				
mesh sizes	$Re = 10^2$	$Re = 5\times10^2$	$Re = 6\times10^2$	$Re = 7\times10^2$	$Re = 8\times10^2$
600×20	91	93	94	99	98
1200×40	118	127	132	134	136

problem. Table 5.9 shows how the number of ASPIN(1) iterations and the average number of GMRES iterations change when we increase the mesh size from a coarse mesh, 600×20, to a fine mesh, 1200×40 on 20 (5×4) processors. We see that the average number of ASPIN(1) iterations remains the same in the case of $Re = 100$. However, if $Re = 800$, the iteration number doubles as we increase the mesh size. The average number of GMRES iterations increases for all values of Reynolds number as expected. In Table 5.10, we observe that increasing the overlapping size can reduce both ASPIN(1) iterations and GMRES iterations when the Reynolds number is high. Table 5.11 shows that the number of ASPIN(1) iterations is not sensitive to the number of processors, while the average GMRES iterations increase a lot when we increase the number of processors from 10 to 20 on a fixed 1200×40 mesh.

Solving the subdomain nonlinear problems In Table 5.12, we compare the numbers of Newton iterations for solving the subdomain nonlinear problems. In this test case, we partition the domain into 5×4 subdomains and number them naturally, first from bottom to top, and then from left to right. See Figure 5.10. Note that the inlet boundary is shared by two subdomains Ω_3 and Ω_4; the

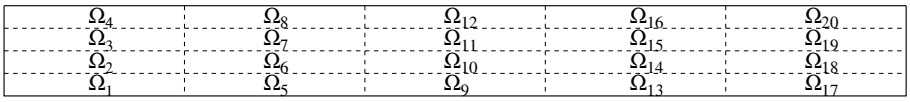
Table 5.10: Backward-facing step problem: Varying the overlapping sizes on a 120×40 mesh.

	Number of ASPIN iterations				
ovlp	$Re = 10^2$	$Re = 5{\times}10^2$	$Re = 6{\times}10^2$	$Re = 7{\times}10^2$	$Re = 8{\times}10^2$
2	5	18	19	23	39
4	5	12	16	15	26
6	5	9	11	12	13
	Average number of GMRES iterations				
ovlp	$Re = 10^2$	$Re = 5{\times}10^2$	$Re = 6{\times}10^2$	$Re = 7{\times}10^2$	$Re = 8{\times}10^2$
2	118	127	132	142	130
4	83	88	90	97	94
6	67	77	80	83	84

Table 5.11: Backward-facing step problem: Different number of processors with same mesh 1200×40.

	Number of ASPIN iterations				
n_p	$Re = 10^2$	$Re = 5{\times}10^2$	$Re = 6{\times}10^2$	$Re = 7{\times}10^2$	$Re = 8{\times}10^2$
$5{\times}2 = 10$	5	12	20	27	35
$5{\times}4 = 20$	5	18	19	28	39
	Average number of GMRES iterations				
n_p	$Re = 10^2$	$Re = 5{\times}10^2$	$Re = 6{\times}10^2$	$Re = 7{\times}10^2$	$Re = 8{\times}10^2$
$5{\times}2 = 10$	62	68	69	71	72
$5{\times}4 = 20$	118	127	132	134	130

Figure 5.10: Backward facing step problem: subdomain numbering



outlet boundary is shared by four subdomains, Ω_{17}, Ω_{18}, Ω_{19}, and Ω_{20}. From Figure 5.11, we observe that for high Reynolds number flows there are two singularities within subdomains from Ω_1 to Ω_4 and the pressure changes significantly in subdomains Ω_3 and Ω_4 compared with others such as the subdomains from Ω_{15} to Ω_{20}. As expected, more Newton iterations are needed in the subdomains Ω_3 and Ω_4, about four times as many as in other smooth regions.

Figure 5.11: Backward-facing step problem: Pressure contours for different values of Reynolds number. The solutions are obtained by ASPIN using a 1200×40 mesh on 5×4 processors.

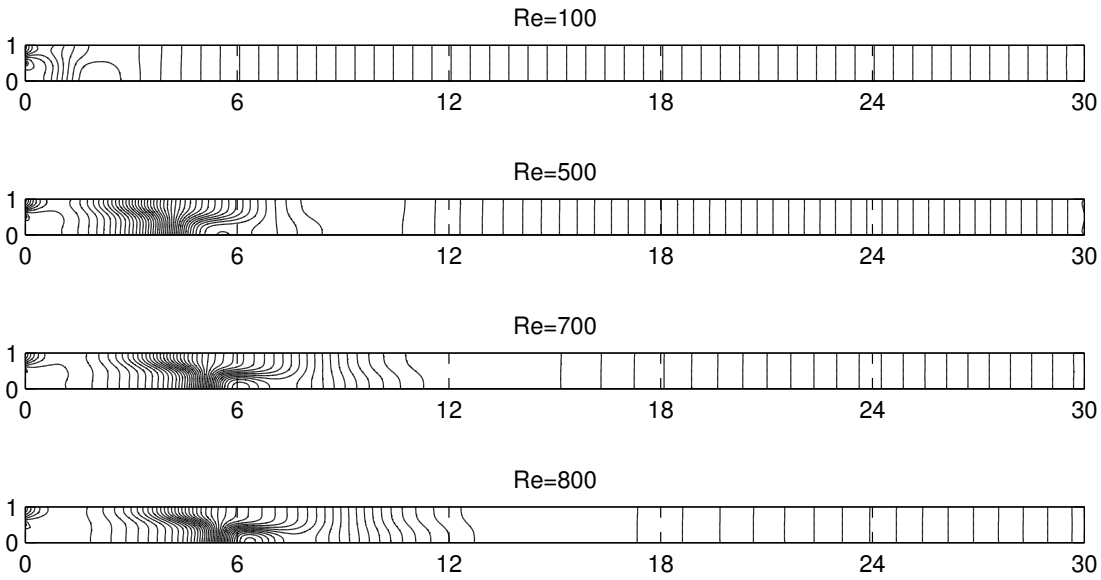


Table 5.12: Backward facing step problem: Total number of subdomain nonlinear iterations. 1200 × 40 mesh, 5 × 4 subdomains on 20 processors.

subdomain	$Re = 100$	$Re = 500$	$Re = 700$	$Re = 800$
Ω_1	8	38	74	109
Ω_2	11	46	85	127
Ω_3	13	61	124	182
Ω_4	13	67	147	238
Ω_5	5	40	78	115
Ω_6	5	43	78	111
Ω_7	5	43	77	109
Ω_8	5	38	78	118
Ω_9	4	18	42	59
Ω_{10}	4	25	45	63
Ω_{11}	4	24	46	71
Ω_{12}	4	18	45	67
Ω_{13}	4	15	26	38
Ω_{14}	4	17	36	54
Ω_{15}	4	15	39	56
Ω_{16}	4	15	26	38
Ω_{17}	6	28	51	74
Ω_{18}	6	29	52	76
Ω_{19}	6	28	55	75
Ω_{20}	6	28	51	74

5.5 Two-level ASPIN for incompressible Navier-Stokes equations

The one-level ASPIN is robust, but not linearly scalable with respect to the number of processors. Some coarse preconditioner is required to couple the subdomain preconditioners. One such coarse preconditioner is proposed and tested in [21] and [79]. The nonlinear coarse system is obtained by the discretization of original nonlinear PDEs on a coarse mesh. Although, in general, solving the coarse systems is easier than the fine systems, a NKS method sometimes is not good enough to converge the coarse system. Therefore, ASPIN(1) is used to solve the coarse system in [21, 79]. To evaluate the coarse function at certain point, one needs to solve a set of nonlinear systems of equations. Although the ASPIN(1) based coarse solver provides good mathematical properties, such as helping speed up the convergence of the linear iterative method, the computational cost to solve many coarse systems is usually high in practice. Numerical experiments [79] show that the ASPIN(1) based coarse solver work fine only for the moderate number of processors, for the large number of processors, a more efficient coarse solver is needed.

5.5.1 A parallel linear coarse component for the nonlinear preconditioner

Here we introduce a new coarse system, which is linear, and the system is constructed by a linearization of the nonlinear coarse system mentioned above, using a Taylor approximation. The coarse function evaluation only requires the solution of a linear system, and hence the computational cost is reduced considerably. More precisely, we assume there exists a finite element mesh $\mathcal{T}^H$ covering the domain Ω. The two meshes $\mathcal{T}^H$ and $\mathcal{T}^h$ do not have to be nested. For the purpose of parallel computing, the coarse mesh is partitioned into non-overlapping

subdomains $\{\Omega_i^H\}$ and overlapping subdomains $\{\Omega_i^{H,\delta}\}$. The corresponding sets of mesh points are denoted by $\{\mathcal{T}_i^H\}$, and $\{\mathcal{T}_i^{H,\delta}\}$. For the simplicity of our software implementation, we assume a non-overlapping partition to be *nested*. In other words, we must have

$$\Omega_i^h = \Omega_i^H$$

for $i = 1, \ldots, N$, even though the corresponding sets of mesh points do not have to be nested; i.e.,

$$\mathcal{T}_i^h \neq \mathcal{T}_i^H.$$

This also means that the same number of processors is used for both the fine and coarse mesh problems. If the overlap is taken into account, in general,

$$\Omega_i^{h,\delta} \neq \Omega_i^{H,\delta}, \quad \text{and} \quad \mathcal{T}_i^{h,\delta} \neq \mathcal{T}_i^{H,\delta}.$$

As in the fine mesh case, we can also define the restriction and extension operators R_i^c and $(R_i^c)^T$ for each coarse subdomain. On the coarse mesh $\mathcal{T}^H$, we can define finite element subspaces similar to the ones defined on the fine meshes, and discretize the original Navier-Stokes equations to obtain a nonlinear system of equations,

$$F^c(x_c^*) = 0 \tag{5.25}$$

We assume that the coarse solution x_c^* of (5.25) is determined through a pre-processing step. We denote J^c as the Jacobian matrix of the coarse mesh function F^c. Similar to the fine mesh, on the coarse subdomains, we obtain the coarse Jacobian submatrices

$$J_i^c = (R_i^c) J^c (R_i^c)^T, i = 1, \ldots, N.$$

We next define the coarse-to-fine and fine-to-coarse grid transfer operators. Let $\{\phi_j^H(x), j = 1, \ldots, m\}$ be the finite element basis functions on the coarse grid,

where m is the total number of coarse grid points in $\mathcal{T}^H$. We define an $3n \times 3m$ matrix I_H^h, the coarse-to-fine extension matrix, as

$$I_H^h = [E_1\, E_2 \cdots E_n]^T,$$

where the block matrix E_i of size $3 \times 3m$ is given by

$$E_i = \begin{bmatrix} (e_H^h)_i & 0 & 0 \\ 0 & (e_H^h)_i & 0 \\ 0 & 0 & (e_H^h)_i \end{bmatrix}$$

and the row vector $(e_H^h)_i$ of length m is given by

$$(e_H^h)_i = \left[\phi_1^H(x_i), \phi_2^H(x_i), \ldots \phi_m^H(x_i)\right],\ x_i \in \mathcal{T}^h$$

for $i = 1, \ldots, n$. A global coarse-to-fine extension operator I_H^h can be defined as the transpose of I_h^H.

To define the coarse function $T_0 : R^{3n} \to R^{3n}$, we introduce a projection $T^c : R^{3n} \to R^{3m}$ as the solution of the linearize coarse system

$$F^c(x_c^*) + J^c(x_c^*)(T^c(x) - x_c^*) = I_h^H F(x), \tag{5.26}$$

for any given $x \in R^{3n}$. Note that the left hand side of (5.26) is a first order Taylor approximation of $F^c(x)$ at the exact coarse mesh solution, x_c^*.

Since $F^c(x_c^*) = 0$, we rewrite (5.26) as

$$T^c(x) = x_c^* + (J^c(x_c^*))^{-1} I_h^H F(x),$$

provided that $J^c(x_c^*)$ is nonsingular. It is easy to see that $T^c(x^*)$ can be computed without knowing the exact solution x^* of F, and $T^c(x^*) = x_c^*$. Then the coarse function can be defined as

$$T_0(x) = I_H^h(T^c(x) - T^c(x^*)) = I_H^h(J^c(x_c^*))^{-1} I_h^H F(x)$$

and its derivative is given by

$$\frac{\partial T_0(x)}{\partial x} = I_H^h (J^c(x_c^*))^{-1} I_h^H J(x). \tag{5.27}$$

We introduce a new nonlinear function

$$\mathcal{F}^{(2)}(x) = T_0(x) + \sum_{i=1}^{N} R_i^T T_i(x),$$

and combining (5.27) and (5.23), we obtain an approximation of Jacobian of $\mathcal{F}^{(2)}$ in the form

$$\widehat{\mathcal{J}}^{(2)}(x) = \left\{ I_H^h (J^c(x_c^*))^{-1} I_h^H + \sum_{i=1}^{N} \left[R_i^T (J_i(x))^{-1} R_i \right] \right\} J(x).$$

The two-level additive Schwarz preconditioned inexact Newton algorithm with a linear coarse solver (ASPIN(2)) is defined as: Find the solution x^* of (3.14) by solving the nonlinearly preconditioned system

$$\mathcal{F}^{(2)}(x) = 0, \tag{5.28}$$

using INB with an initial guess $x^{(0)}$. Details of ASPIN(2) is given below.

5.5.2 Details of the two-level ASPIN

Let $x^{(0)}$ be an initial guess and $x^{(k)}$ the current approximate solution. Then a new approximate solution $x^{(k+1)}$ can be computed by ASPIN(2) algorithm as follows:

Algorithm 3 ((ASPIN(2)) .

Step 1: Evaluate the nonlinear residual $\mathcal{F}^{(2)}(x)$ at $x^{(k)}$ through the following steps:

(1) Find $w_0^{(k)}$ by solving the linearize coarse mesh problem

$$J^c(x_c^*) z_c = I_h^H F(x^{(k)}) \tag{5.29}$$

using a Krylov-Schwarz method with a left preconditioner, $P^{-1} = \sum_{i=1}^{N} (R_i^c)^T (J_i^c)^{-1} R_i^c$ and the initial guess $z_c = 0$.

(2) Find $w_i^{(k)} = T_i(x^{(k)})$ by solving in parallel, the local nonlinear systems

$$G_i(w) \equiv F_i(x^{(k)} - R_i^T w) = 0 \tag{5.30}$$

using Newton method with backtracking and the initial guess $w = 0$.

(3) Form the global residual

$$\mathcal{F}^{(2)}(x^{(k)}) = I_H^h w_0^{(k)} + \sum_{i=1}^{N} R_i^T w_i^{(k)}.$$

Step 2: Check the stopping condition on $||\mathcal{F}^{(2)}(x^{(k)})||_2$. If $||\mathcal{F}^{(2)}(x^{(k)})||_2$ is small enough, stop, otherwise, continue.

Step 3: Evaluate pieces of the Jacobian matrix $\widehat{\mathcal{J}}^{(2)}(x)$ of the preconditioned system that are needed in order to multiply (5.31) below with a vector in the next step. This includes $J(x^{(k)})$ as well as J_i and its sparse LU factorization.

$$\widehat{\mathcal{J}}^{(2)} = \left\{ I_H^h (J^c(x_c^*))^{-1} I_h^H + \sum_{i=1}^{N} \left[R_i^T (J_i(x^{(k)}))^{-1} R_i \right] \right\} J(x^{(k)}). \tag{5.31}$$

Step 4: Find an inexact Newton direction $s^{(k)}$ by solving the following Jacobian system approximately using a Krylov subspace method

$$\widehat{\mathcal{J}}^{(2)} s^{(k)} = -\mathcal{F}^{(2)}(x^{(k)}) \tag{5.32}$$

in the sense that

$$||\mathcal{F}^{(2)}(x^{(k)}) + \widehat{\mathcal{J}}(x^{(k)}) s^{(k)}||_2 \leq \eta_k ||\mathcal{F}^{(2)}(x^{(k)})||_2 \tag{5.33}$$

for some $\eta_k \in [0, \eta_{max}]$ for some $\eta_{max} < 1$ independent of k.

Step 5: Scale the search direction $s^{(k)} \leftarrow \frac{s_{max}}{||s^{(k)}||_2} s^{(k)}$ if $||s^{(k)}||_2 \geq s_{max}$.

Step 6: Compute a new approximate solution

$$x^{(k+1)} = x^{(k)} + \lambda^{(k)} s^{(k)},$$

where $\lambda^{(k)}$ is a damping parameter determined by the standard backtracking procedure.

Remark 10 No preconditioning is used in Step 4 of Algorithm 3. In fact, $\widehat{\mathcal{J}}^{(2)}$ can be viewed as the original Jacobian system J preconditioned by a two-level additive Schwarz preconditioner, where the coarse mesh preconditioner $I_H^h(J^c(I_H^h x^{(k)} I_h^H)^{-1} I_h^H$ is approximated by $I_H^h(J^c(x_c^*))^{-1} I_h^H$. Hence, $\widehat{\mathcal{J}}^{(2)}$ is well-conditioned through nonlinear preconditioning as long as the $I_H^h x^{(k)} I_h^H$ is close to x_c^*.

Remark 11 Similar to the one-level method, if a Krylov subspace method is used to the Jacobian problem, only the Jacobian-vector product, $u = \widehat{\mathcal{J}}^{(2)} v$, is required. In a distributed-memory parallel implementation, this operation consists of five phrases:

(1) Solve $J^c(x_c^*) z_c = I_h^H v$, using a Krylov-Schwarz method with a left preconditioner, $P^{-1} = \sum_{i=1}^N (R_i^c)^T (J_i^c)^{-1} R_i^c$ and the initial guess $z_c = 0$.

(2) Perform the matrix-vector multiply, $w = Jv$, in parallel.

(3) On each subdomain, collect the data from the subdomain and its neighboring subdomains, $w_i = R_i w$.

(4) Solve $J_i u_i = w_i$ using a sparse direct solver.

(5) Send the partial solutions and its neighboring subdomain and take the sum, $u = \sum_{i=1}^N R_i^T u_i + I_H^h z_c$.

5.6 Numerical results: II.

5.6.1 Comparison one-level ASPIN and two-level ASPIN

In this section, we study the performance of one-level and two-level ASPIN. Particularly, we study the linear and nonlinear scalability of the methods. Only

machine independent results are reported. At the fine mesh level, we use the cubic line search technique [29] as described in subsection 5.2.1 for both global and local nonlinear problems. The global nonlinear iteration is stopped if the condition $||\mathcal{F}^{(2)}(x^{(k)})||_2 \leq 10^{-6}||\mathcal{F}^{(2)}(x^{(0)})||_2$ is satisfied, and the local nonlinear iteration on each subdomain is stopped if the condition $||G_i(w_{i,l}^{(k)})||_2 \leq 10^{-4}||G_i(w_{i,0}^{(k)})||_2$ is satisfied. We use a restarted GMRES(200) for solving the global Jacobian systems (5.32). The global linear iteration is stopped if the relative tolerance $||\mathcal{F}^{(2)}(x^{(k)}) + \mathcal{J}^{(2)}(x^{(k)})s^{(k)}||_2 \leq 10^{-6}||\mathcal{F}^{(2)}(x^{(k)})||_2$ is satisfied. During local nonlinear iterations, a direct sparse solver, LU decomposition, is employed for solving each local Jacobian system. At the coarse mesh level, we use a restarted GMRES(200) method with a left overlapping Schwarz preconditioner to solve the coarse systems (5.29). The stopping criterion for the coarse mesh problem is that the condition $||I_h^H F(x^{(k)}) - J^c(x_c^*)z_c||_2 \leq 10^{-10}||I_h^H F(x^{(k)})||_2$ is satisfied. We use the overlapping size, $\delta = 2$ for both the fine and coarse systems. As suggested in [60], we include the re-scaling of the search direction $s^{(k)}$ in Step 5 if $||s^{(k)}||_2 \geq s_{max}$ to enhance the robustness of ASPIN for solving incompressible flows. This step also reduces the number of line search steps since the evaluation of nonlinearly preconditioned function is expensive. All numerical results reported here are based on the optimal choice of the parameter s_{max} which results in the smallest number of global nonlinear iterations.

We first study the effect of the coarse mesh size on the global nonlinear iterations and the global linear iterations of ASPIN(2) for different values of Reynolds number. In this set of numerical experiments, all results are obtained using a fixed fine mesh 128×128 on 16 processors, and the coarse mesh size is varied from 16×16 to 80×80. Table 5.13 shows that to apply two-level methods on the moderate number of processors, the coarse mesh has to be sufficiently fine, say 40×40 in this case. For this particular case, the numbers of global nonlinear iter-

ations, as well as global linear iterations, are not very sensitive with the increase of Reynolds number.

To study the parallel scalability of ASPIN(2) with respect to the number of processors, we use a fixed fine mesh 128×128 and a coarse mesh 40×40. For comparison purposes, we also include the results obtained using ASPIN(1). Table 5.14 shows that by adding a coarse preconditioner, not only the global linear iterations is reduced significantly as we increase the number of processors from 4 to 64, but also the global nonlinear iterations is improved especially for high Reynolds number flows. Table 5.15 shows the mesh scalability of ASPIN(2). Now we increase the fine mesh size from 128×128 to 512×512 on 64 processors, and we use a fixed coarse mesh 64×64 for all test cases. We see that the coarse mesh of size 64×64 works fine only for first two small meshes (128×128 and 256×256). The global linear iterations increase quite a bit for the largest mesh (512×512). For a fully scalable algorithm, one may need to increase the coarse mesh size as well, when the fine mesh size and the number of processors increase. The optimal coarse mesh size also depends mildly on the Reynolds number.

In the next set of numerical experiments, instead of using a zero vector as the initial guess, we employ the interpolation of the coarse mesh solution, $I_H^h x_c^*$, as the initial guess. Here we refer to the two-level ASPIN with the new initial guess as ASPIN(2'). Note that no additional cost is needed for this new selection of the initial guess since we need x_c^* for the nonlinear function evaluation anyway. However, with this small modification, we see a much better overall performance of the two-level ASPIN. We rerun the same test cases presented in Table 5.15 using ASPIN(2') and the results are shown in Table 5.16. Obviously ASPIN(2') performs better than ASPIN(2). ASPIN(2') requires only about half of the global nonlinear and linear iterations than ASPIN(2).

Table 5.13: ASPIN(2): Varying the coarse mesh size for different values of Reynolds number. Fine mesh: 128×128. The number of processors n_p=16.

Coarse mesh	$Re = 10^3$	3×10^3	5×10^3	8×10^3	10^4
Number of global nonlinear iterations					
16×16	8	11	11	14	17
20×20	9	9	11	13	14
32×32	8	9	11	10	12
40×40	8	9	9	10	11
64×64	8	10	9	11	11
Average number of global linear iterations					
16×16	58	74	94	111	122
20×20	50	66	75	89	103
32×32	45	52	59	64	68
40×40	43	50	54	60	60
64×64	42	49	52	55	65

Table 5.14: ASPIN(1) and ASPIN(2): Varying the number of processors. Fine mesh size: 128×128. Coarse grid size: 40×40.

n_p	$Re = 10^3$	3×10^3	5×10^3	8×10^3	10^4
ASPIN(1)					
Number of global nonlinear iterations					
$2 \times 2 = 4$	9	10	13	19	19
$4 \times 4 = 16$	9	12	12	16	18
$8 \times 8 = 64$	10	15	14	19	19
Average number of global linear iterations					
$2 \times 2 = 4$	67	69	71	73	74
$4 \times 4 = 16$	127	128	133	137	140
$8 \times 8 = 64$	395	394	400	497	655
ASPIN(2)					
Number of global nonlinear iterations					
$2 \times 2 = 4$	9	9	11	10	12
$4 \times 4 = 16$	8	9	9	10	11
$8 \times 8 = 64$	8	9	12	12	14
Average number of global linear iterations					
$2 \times 2 = 4$	33	40	40	40	46
$4 \times 4 = 16$	43	50	54	60	60
$8 \times 8 = 64$	49	62	61	78	79

Table 5.15: ASPIN(2): Varying the fine mesh size for different values of Reynolds number. Coarse mesh size: 64×64. The number of processors $np = 64$

Fine mesh	$Re = 10^3$	3×10^3	5×10^3	8×10^3	10^4
Number of global nonlinear iterations					
128×128	8	9	10	11	10
256×256	8	11	12	13	14
512×512	9	11	14	15	19
Average number of global linear iterations					
128×128	50	61	70	87	94
256×256	63	82	85	93	94
512×512	90	134	171	144	111

Table 5.16: ASPIN(2'): Varying the fine mesh size for different values of Reynolds number. Coarse mesh size: 64×64. The number of processor $np = 64$.

Fine mesh	$Re = 10^3$	3×10^3	5×10^3	8×10^3	10^4
Number of global nonlinear iterations					
128×128	3	4	5	5	6
256×256	3	4	5	6	7
512×512	4	5	6	7	8
Average number of global linear iterations					
128×128	26	28	30	35	36
256×256	31	34	38	43	48
512×512	48	49	51	56	60

5.6.2 Preliminary timing results

Since our code has yet not been optimized, here, we report some preliminary timing results to provide some insights into future improvement. First, we keep work load per processor roughly the same and simultaneously increase the number of processors from 4 to 64 and the problem size from 128 by 128 to 512 by 512. As shown Table 5.17, ASPIN(2') is nearly optimal for a moderate number of processors, since the CPU time per global nonlinear iterations do not grow with the number of processors and the problem size.

Table 5.18 and 5.19 presents the breakdown of the timing results and iteration counts for the components in the ASPIN(2'): the global function evaluations, including the coarse mesh part in (5.29) and local subdomain part in (5.30), the construction of the Jacobian J of the original function, which is needed in (5.31), and the solution of the coarse-mesh part in Step 1 in Remark 10.

As expected, the most time-consuming component in ASPIN(2') is the local subdomain part of global-function evaluation, taking over 60% of total time for each test case, since each processor needs to solve several nonlinear subdomain systems. Exact LU decomposition for solving the Jacobian systems and the construction of the Jacobian matrices using multi-colored finite differences take a lot of time. However, these two operations are purely local and do not involve any communications. On the other hand, the *linear* coarse mesh function evaluation is very efficient less than 1% of the total time.

From the tables, we also find that the only non-scalable component in ASPIN(2') is the coarse parts of the Jacobian matrices, $J^c(x_c^*)$. It is expected the CPU time of this component will be dominant for the large number of processors. Replacing a parallel iterative coarse solver by a parallel direct coarse solver may be a solution. Since it is fixed through each global nonlinear iterations as

Table 5.17: Parallel scalability of ASPIN(2')

Case	Fine/Coarse mesh	n_p	Global nonlinear	Time	Time/nonlinear its.
1	128/32	4	14	1385	99
2	256/64	16	9	1007	112
3	512/64	64	9	965	107

well as global linear iterations, we can take an advantage of that and a parallel direct solver is a good candidate for solving such linear system. We decompose the matrix once at the beginning of the iterations, and store the upper and lower triangulars of the matrix in . Then we perform forward and backward substitution in parallel at every iterations. It is expected that ASPIN(2') based on a parallel coarse direct solver will outperform the one based on a parallel iterative solver. However, the performance of a direct solver depends heavily on the ordering and pivoting strategies. How these factors affect overall performance of ASPIN(2') need to be investigated further.

5.7 Concluding remarks

Finding a fast, robust and scalable solver for incompressible Navier-Stokes equations is one of the key research areas in computational fluid dynamics. In this chapter, we developed a fully parallel nonlinearly preconditioned inexact Newton method for solving incompressible Navier-Stokes equations in the primitive vari-

Table 5.18: Number of iterations: Break down of each component in the ASPIN(2')

Case	Global nonlinear	Global linear	AVE Coarse ite
1	14	34	44
2	9	33	115
3	9	46	194

Table 5.19: Timing: Break down of each component in the ASPIN(2')

	Function evaluation				
Case	Coarse	Subdomains	Form J	Solve J^c	Total
1	8	986	300	55	1385
2	9	608	240	127	1007
3	8	551	223	152	965

able form. The one-level nonlinear preconditioner is constructed using the overlapping additive Schwarz domain decomposition method. A PETSc based parallel software package was developed and tested for the two-dimensional incompressible Navier-Stokes equations discretized with a stabilized $Q_1 - Q_1$ finite element method. From the numerical experiments on two benchmark problems including a lid-driven cavity flow problem and a backward-facing step problem, we concluded that the new method is more robust than the commonly used inexact Newton method with backtracking for high Reynolds number flows.

To enhance the parallel scalability, we also introduced a new two-level ASPIN algorithm, and constructed the two-level nonlinear preconditioner using a local nonlinear overlapping Schwarz domain decomposition and a global linear coarse solver. We obtained some encouraging numerical results for a moderate number of processors. We show that the new two-level ASPIN maintains the fast convergence and robustness properties of the one-level ASPIN. In addition, if the coarse-mesh size is fine enough, the new algorithms provide better nonlinear and linear scalability with respect to the number of processors.

Chapter 6

Conclusions and Further Work

Because the numerical solution of large, sparse linear and nonlinear systems of equations arising from the discretization of PDEs is required in many computational science and engineering applications, finding parallel, fast, scalable, robust algorithms for solving such systems is currently one of the most active areas of research in computational mathematics. Here, our recommendations of some strategies and some further research for solving general large nonlinear systems of equations are summarized as follows.

Since the global function evaluations of ASPIN are much more expensive than these of NKS, ASPIN is intended for cases in which NKS fails to convergence or experiences unacceptably slow convergence. If INB is robust enough for a large set of problem parameters, NKS is always a first choice. From the viewpoint of implementation, the one-level NKS is preferable to multi-level NKS, since finding an optimal of coarse-mesh sizes may be difficult, and multiple meshes may increase complexity of parallel implementation, especially for unstructured meshes. However, from the theoretical analysis or numerical experience, we learn that convergence rates of one-level Schwarz methods degrade as the number of processors increases. This situation gets worse for the cases of steady-state. Therefore, two-level Schwarz methods for solving the Jacobian systems are recommended. On the other hand, in the cases of unsteady state the two questions need to be answered–

whether nonlinear preconditioning is required for time-dependent problems, or NKS already robust enough for a large range of problem parameters, and whether a coarse mesh solver is required, or an implicit setting, e.g. backward Euler time integration, is sufficient for scalability of one-level methods. Recent studies of the bidomain model for simulating the electrical activity of the heart [80] and the unsteady diffusion-radiation problem [82] showed that NKS is robust since the numerical solution at the previous time step is a good initial guess for INB. Furthermore, although the number of linear iterations for one-level NKS increases slightly as the number of processors increases, the overall CPU time is completely scalable up to 128 processors. In this dissertation, we restrict our discussion to the steady-State Navier-Stokes equations. Due to the nature of the fully coupled approach, the extensions of our approaches to time-dependent cases or to the Navier-Stokes equations coupled with other physical equations are straightforward. While most of the literature is concerned with the stability and accuracy of a variety of time-integration schemes [31, 100], we need to further investigate some problems related to linear and nonlinear solvers to obtain some inclusive conclusions.

If the robustness becomes an issue, there are some techniques available to enhance the robustness of INB. To summarize, we have:

(1) INB robustness parameters: In some certain applications, the adaptive forcing terms η_k, suggested by Eisenstat and Walker [42], are able to improve the robustness of INB. In addition, as introduced by Dennis and Schnabel in the page 129 of [29], the re-scaling parameter s_{max}, which plays an important role to enhance robustness of ASPIN for solving incompressible flows, may be helpful in the cases of INB as well.

(2) Continuation approaches, including parameter continuation [2, 53], mesh

sequencing [75], and pseudo time stepping [25, 67, 73].

(3) Nonlinear preconditioning: The key idea of nonlinear preconditioning is to reformulate an ill-conditioned system to a well-conditioned system without changing the solution. In this dissertation, we propose and test several parallel multilevel linear and nonlinear Schwarz methods for the Stokes problem and incompressible Navier-Stokes equations. The techniques of *nonlinear* preconditioning based on the Schwarz framework are general so that they can be applied to other types of problem easily. In addition, the techniques are very powerful, not only for enhancing the robustness of some iterative methods but also for improving the scalability of these methods. For example, through nonlinear preconditioning by one-level ASPIN, we have:

- Robustness:
  - an improved angle estimate

$$\frac{1}{\kappa(\left\{\sum_{i=1}^{N} R_i^T (J_i)^{-1} R_i\right\} J)} \leq \cos(\theta); \text{ and}$$

  - an improved merit function $\|\mathcal{F}\|^2/2$ for the line search
- Scalability:
  - an improved conditioning of the Jacobian system

$$\left(\left\{\sum_{i=1}^{N} R_i^T (J_i)^{-1} R_i\right\} J\right) s^{(k)} = -\mathcal{F}^{(2)}(x^{(k)}).$$

To show the applicability of ASPIN to larger problems, more applications with complex geometry need to be tested using larger numbers of processors in the future. In particular, we are interested in studying the efficiency of applying the three different strategies to solving linear coarse

mesh problems– a parallel preconditioned iterative method as described in Chapter 5; a parallel direct solver, such as SuperLU_dist [28]; and a sequential direct solver applied to solving the coarse mesh problem on each processor redundantly, as well as affects of all three strategies on the overall performance of ASPIN on massively parallel computers. We would also like to explore possibilities for other applications of ASPIN. Our target applications are those problems that are difficult to solve by INB or other fully decoupled methods, such as a SIMPLE-like algorithm, for example, drift-diffusion equations in semiconductor device simulations [23, 24, 103] and coupled fluid-structure interaction problems [48].

Finally, it is should be noted that due to the fact of complexity of nonlinear systems, the interplay of three strategies discussed above is often needed for a parallel algorithm to achieve optimal. For example, ASPIN(2') introduced in Chapter 5 combines all three strategies: the well-selected re-scaling parameters $s_{\max}$, the usage of mesh sequencing, where the interpolation of the solution on coarse meshes is used as initial guess for INB, and nonlinear preconditioning based on the Schwarz framework.

Bibliography

[1] R. Aitbayev, X. -C. Cai, and M. Paraschivoiu, Parallel two-level methods for three-dimensional transonic compressible flow simulations on unstructured meshes, in Parallel Computational Fluid Dynamics: Towards Teraflops, Optimization and Novel Formulations, D. E. Keyes et al. Eds., Elsevier, Amsterdam, 2000, pp. 89-96.

[2] V. F. DE Almeida and J. J. Derby, Construction of solution curves for large two-dimensional problems of steady-state flows of incompressible fluids, SIAM J. Sci. Comput., 22 (2000), pp. 285-311.

[3] P. R. Amestoy, I. S. Duff, J.-Y. LExcellent, and J. Koster, MUltifrontal Massively Parallel Solver (MUMPS Version 4.3) Users guide, 2003.

[4] W. K. Anderson, W. D. Gropp, D. K. Kaushik, D. E. Keyes and B. F. Smith, Achieving High Sustained Performance in an Unstructured Mesh CFD Application, in Proceedings of Supercomputing 99, IEEE Computer Society, 1999.

[5] C. Ashcraft, D. Perece, D. K. Wah, and J Wu, The reference mannual for SPOOLES: an object oriented software library for solving sparse linear systems of equations, Boeing Phantom Works, 1999.

[6] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, PETSc Users Manual, ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2002.

[7] G. Biros and O. Ghattas, Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization, Part I: the Krylov-Schur solver, SIAM J. Sci. Comput., to appear.

[8] G. Biros and O. Ghattas, Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization, Part II: the Lagrange-Newton solver and its application to optimal control of steady viscous flows, SIAM J. Sci. Comput., to appear.

[9] C. Bischof, A. Carle, P. Hovland, P. Khademi, A. Mauer ADIFOR 2.0 User's Guide, Argonne National Laboratory, 1998.

[10] J. H. Bramble and J. E. Pasciak, A domain decomposition technique for Stokes problems, Appl. Numer. Math., 6 (1990), pp. 251-261.

[11] J. H. Bramble, J. E. Pasciak, and A. T. Vassilev, Analysis of the inexact Uzawa algorithm for saddle point problems, SIAM J. Numer. Anal., 34 (1997), pp. 1072-1092.

[12] A. N. Brooks and T. J. R. Hughes, Streamline upwind/Petrov-Galerkin formulations for convective dominated flows with particular emphasis in the incompressible Navier-Stokes equations, Comput. Methods Appl. Mech. Engrg., 32 (1982), pp. 199-259.

[13] X. -C. Cai, Additive Schwarz algorithms for parabolic convection-diffusion equations, Numer. Math., 60 (1990), pp. 41-62.

[14] X. -C. Cai and O. Widlund, Domain decomposition algorithms for indefinite elliptic problems, SIAM J. Sci. Stat. Comp., 13 (1992), pp. 243-258.

[15] X. -C. Cai, An optimal two-level overlapping domain decomposition method for elliptic problems in two and three dimensions, SIAM J. Sci. Comput., 14 (1993), pp. 239-247.

[16] X. -C. Cai A family of overlapping Schwarz algorithms for nonsymmetric and indefinite elliptic problems, in Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering, D. E. Keyes, et al. Eds., SIAM, Philadelphia, PA, 1994.

[17] X. -C. Cai, Multiplicative Schwarz methods for parabolic problems, SIAM J. Sci. Comput., 15 (1994), pp. 587-603.

[18] X. -C. Cai, W. D. Gropp, D. E. Keyes, A comparison of some domain decomposition and ILU preconditioned iterative methods for nonsymmetric elliptic problems, Numer. Lin. Alg. Applics., 1 (1994), pp. 477-504.

[19] X. -C. Cai, W. D. Gropp, D. E. Keyes, R. G. Melvin, and D. P. Young, Parallel Newton-Krylov-Schwarz algorithms for the transonic full potential equation, SIAM J. Sci. Comput., 19 (1998), pp. 246-265.

[20] X. -C. Cai and D. E. Keyes, Nonlinearly preconditioned inexact Newton algorithms, SIAM J. Sci. Comput., 24 (2002), pp. 183-200.

[21] X. -C. Cai, D. E. Keyes, and L. Marcinkowski, Nonlinear additive Schwarz preconditioners and applications in computational fluid dynamics, Int. J. Numer. Meth. Fluids, 40 (2002), pp. 1463-1470.

[22] X. -C. Cai, D. E. Keyes, and D. P. Young, A nonlinear additive Schwarz preconditioned inexact Newton method for shocked dust flows, in Proceedings of the 13th International Conference on Domain Decomposition Methods, CIMNE, Barcelona, Spain, 2002.

[23] R. -C. Chen and J. -L. Liu, An iterative method for adaptive finite element solutions of an energy transport model of semiconductor device, J. Comp. Phys., 189 (2003), pp. 579-606.

[24] R. -C. Chen and J. -L. Liu, Monotone iterative methods for the adaptive finite element solution of semiconductor equations, J. Comp. and Applied Math., 159 (2003), pp. 341-364.

[25] T. S. Coffey, C. T. Kelley, and D. E. Keyes, Pseudo-transient continuation and differential-algebraic equations, SIAM J. Sci. Comput., 25 (2003), pp. 553-569.

[26] T. F. Coleman and J. J. Moré, Estimation of sparse Jacobian matrices and graph coloring problem, SIAM J. Numer. Anal., 20 (1983), pp. 243-209.

[27] I. G. Currie, Fundamental Mechanics of Fluids, McGraw-Hill, New York, NY, 1993.

[28] J. W. Demmel, J. R. Gilbert, and X. S. Li, SuperLU Users' Guide, Lawrence Berkeley National Laboratory, 2003.

[29] J. Dennis and R. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, SIAM, Philadelphia, PA, 1996.

[30] R. S. Dembo, Eisenstat, and T. Steihaug, Inexact Newton methods, SIAM J. Numer. Anal., 19 (1982), pp. 400-408.

[31] W. Dettmer, D. Perić, An analysis of the time integration algorithms for the finite element solutions of incompressible NavierStokes equations based on a stabilised formulation, Comput. Methods Appl. Mech. Engrg., 192 (2003), pp. 11771226

[32] P. Deuflhard, Global inexact Newton methods for very large scale nonlinear problems, IMPACT Comp. Sci. Eng., 3 (1991), pp. 366-393.

[33] J. Dongarra, I. Duff, D. Sorensen, and H. van der Vorst, Numerical Linear Algebra for High-Performance Computers, SIAM, Philadelphia, PA, 1998.

[34] J. Douglas and J. Wang, An absolutely stablized finite element for the Stokes problem, Math. Comp. 52 (1989), pp. 495-508.

[35] M. Dryja and O. B. Widlund, An additive variant of the Schwarz alternating method for the case of many subregions, Tech. Report 339, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, (1987).

[36] M. Dryja and O. B. Widlund, Domain decomposition algorithms with small overlap, SIAM J. Sci. Comput., 15 (1994), pp. 604-620.

[37] M. Dryja and W. Hackbusch, On the nonlinear domain decomposition method, BIT 37 (1997), pp. 296-311.

[38] M. Dumett, P. Vassilevski, and C. S. Woodward, A multigrid method for nonlinear unstructured finite element elliptic equations, SIAM J. on Sci. Comput., to appear.

[39] H. C. Elman and G. H. Golub, Inexact and preconditioned Uzawa algorithm for saddle point problems, SIAM J. Numer. Anal., 31 (1994), pp. 1645-1661.

[40] H. C. Elman, Multigrid and Krylov subspace methods for the discrete Stokes equations, Int. J. Numer. Meth. Fluids, 22 (1996), pp. 755-770.

[41] S. C. Eisenstat and H. F. Walker, Globally convergent inexact Newton method, SIAM J. Optim., 4 (1994), pp. 393-422.

[42] S. C. Eisenstat and H. F. Walker, Choosing the forcing terms in an inexact Newton method, SIAM J. Sci. Comput., 17 (1996), pp. 16-32.

[43] H. C. Elman and D. Silvester, Fast nonsymmetric iterations and preconditioning for Navier-Stokes equations, SIAM J. Sci. Comp., 17 (1996), pp. 33-46.

[44] H. C. Elman, V. E. Howle, J. N. Shadid, R. S. Tuminaro, A parallel block multi-level preconditioner for the 3D incompressible Navier-Stokes equations, J. Comput. Phy., 187 (2003), pp. 504-523.

[45] L. P. Franca, S. L. Frey, and T. J. R. Hughes, Stablized finite element method: I. Application to the advective-diffusive model, Comput. Methods Appl. Mech. Engrg., 95 (1992), pp. 253-276.

[46] L. P. Franca and S. L. Frey, Stabilized finite element method: II. The incompressible Navier-Stokes equation, Comput. Methods Appl. Mech. Engrg., 99 (1992), pp. 209-233.

[47] D. K. Gartling, A test problem for outflow boundary conditions- flow over a backward-facing step, Int. J. Numer. Meth. Fluids, 11 (1990), pp. 953-967.

[48] O. Ghattas and X. Li, A variational finite element method for stationary nonlinear fluid-solid interaction, J. Comp. Phys., 121 (1995), pp. 347-356.

[49] U. Ghia, K. N. Ghia, and C. T. Shin, High-Re solution for incompressible flow using the Navier- stokes equations and the multigrid method, J. Comput. Phy., 48 (1982), pp. 387-411.

[50] P. M. Gresho, D. K. Gartling, J. R. Torczynski, K. A. Cliffe, K. H. Winters, T. J. Garratt, A. Spence, and J. W. Goodrich, Is the steady viscous incompressible two-dimensional flow over a backward facing step at Re=800 stable, Int. J. Numer. Meth. Fluids, 17 (1993), pp. 501-541.

[51] M.J. Grote and T. Huckle, Parallel preconditioning with sparse approximate inverses, SIAM J. of Scient. Comput. 18 (1997), pp. 838-853.

[52] M. D. Gunzburger, Finite Element Methods for Viscous Incompressible Flows, Academics Press, New York, NY, 1989.

[53] M. D. Gunzburger and J. Peterson, Predictor and steplength selection in continuation methods for the Navier-Stokes equations, Comput. and Math. Appl., 22 (1991), pp. 73-81.

[54] M. D. Gunzburger, Perspectives in Flow Control and Optimization, SIAM, Philadelphia, PA, 2003.

[55] S. K. Hannani, M. Stanislas, and P. Dupont, Incompressible Navier-Stokes computation with SUPG and GLS formulations – a comparison study, Comput. Methods Appl. Mech. Engrg., 124 (1995), pp. 153-170.

[56] D. Hendriana and L. J. Bethe, On upwind methods for parabolic finite elements in incompressible flows, Int. J. Numer. Meth. Engng., 47 (2000), pp. 317-340.

[57] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers, ACM Trans. Math. Softw., submitted.

[58] F. -N. Hwang and X. -C. Cai, A parallel nonlinear additive Schwarz preconditioned inexact Newton algorithm for incompressible Navier-Stokes equations, J. Comput. Phys., submitted.

[59] F. -N. Hwang and X. -C. Cai, Parallel Schwarz Type Preconditioners for the Stokes Problem, Comput. Methods Appl. Mech. Engrg., submitted.

[60] F. -N. Hwang and X. -C. Cai, Improving robustness and parallel scalability of Newton method through nonlinear preconditioning, in Proceedings of the 15th International Conference on Domain Decomposition Methods, R. Kornhuber et al. Eds., Springer-Verlag, New York, NY, 2004, pp. 201-208.

[61] F. -N. Hwang and X. -C. Cai, Reducing nonlinear complexity of two-level nonlinear additive Schwarz preconditioned inexact Newton method, in preparation.

[62] V. John, A comparison of parallel solvers for the incompressible Navier-Stokes equations, Comput. Visual. Sci., 1 (1999), pp. 193-200.

[63] V. John and L. Tobiska, Numerical performance of smoothers in coupled multgrid for the parallel solution of the incompressible Navier-Stokes equations, Int. J. Numer. Meth. Fluids, 33 (2000), pp. 453-473.

[64] V. John, G. Matthies, T.I. Mitkova, L. Tobiska, P.S. Vassilevski, A Comparison of Three Solvers for the Incompressible Navier-Stokes Equations, in Large-Scle Scientific Computations of Engineering and Environmental Problems II, M. Griebel et al. Eds., Vieweg-Verlag, NY, 2000, pp. 215-222.

[65] C. Johnson, Numerical Solution of Partial Differential Equations by the Finite Element Method, Cambridge University Press, Cambriage, 1987.

[66] G. Karypis, K. Schloegel and V. Kumar, Parallel Graph Partitioning and Sparse Matrix Ordering Library, University of Minnesota, Department of Computer Science and Engineering Army HPC Research Center, 2003.

[67] C. T. Kelley and D. E. Keyes, Convergence analysis of pseudo-transient continuation, SIAM J. Sci. Comput., 35 (1998), pp. 508-523.

[68] D. E. Keyes, How Scalable is Domain Decomposition in Practice?, Proceedings of the 11th Intl. Conf. on Domain Decomposition Methods (C.-H. Lai, et al, eds.), (1998) 286-297.

[69] A. Klawonn, Block-triangular preconditioners for saddle point problems with a penalty term, SIAM J. Sci. Comp., 19 (1998), pp. 172-184.

[70] A. Klawonn, An optimal preconditioner for a class of saddle point problems with a penalty term, SIAM J. Sci. Comput., 19 (1998), pp. 540-552.

[71] A. Klawonn and L. F. Pavarino, Overlapping Schwarz methods for mixed linear elasticity and Stokes problems, Comput. Methods Appl. Meth. Engrg., 165 (1998), pp. 233-245.

[72] A. Klawonn and L. F. Pavarino, A comparison of overlapping Schwarz methods and block preconditioners for saddle point problems, Numer. Linear Algebra Appl., 7 (2000), pp. 1-25.

[73] D. A. Knoll and D. E. Keyes, Jacobian-free Newton-Krylov methods: a survey of approaches and applications, J. Comp. Phys., 193 (2004), pp. 357-397.

[74] P.J. Lanzron, D. J. Rose, J. T. Wilkes, An analysis of approximate nonlinear elimination, SIAM J. Sci. Comput., 17 (1996), pp. 538-559.

[75] W. Layton, H. K. Lee, and J. Peterson, Numerical solution of the stationary Navier–Stokes equations using a multilevel finite element method, SIAM J. Sci. Comput., 20 (1998), pp. 1-12.

[76] J. Li, Dual-primal FETI methods for incompressible Stokes and linearized Navier-Stokes equations, Tech. Report 830, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, (2002).

[77] P. L. Lions, Interprétation stochastique de la methode alternée de Schwarz, R. Acad. Sci. Paris, 268 (1978), pp. 325-328.

[78] J. Mandel, Hybrid domain decomposition with unstructured subdomains, Contemporary Math., 157 (1994), pp. 103-112.

[79] L. Marcinkowski and X. -C. Cai, Parallel performance of some two-level ASPIN algorithms, in Proceedings of the 15th International Conference on Domain Decomposition Methods, R. Kornhuber et al. Eds., Springer-Verlag, New York, NY, 2004, pp. 639-646.

[80] M. Murillo and X. C. Cai, A fully implicit parallel algorithm for simulating the nonlinear electrical activity of the heart, Numer. Linear Algebra Appl. (2004), 11 (2004), pp. 261-277.

[81] J. Nocedal and S. J. Wright, Numerical Optimization, Springer-Verlag, New York, NY, 1999.

[82] S. Ovtchinnikov and X. -C. Cai, One-level Newton-Krylov-Schwarz algorithm for unsteady non-linear radiation diffusion problem, Numer. Linear Algebra Appl. (2004), in press.

[83] S. V. Patanker, Numerical Heat Transfer and Fluid Flow, McGraw-Hill, New York, NY, 1980.

[84] L. F. Pavarino, Indefinite overlapping Schwarz methods for time-dependent Stokes problems, Comput. Methods Appl. Mech. Engrg., 187 (2000), pp. 35-51.

[85] M. Prenice and M. D. Tocci, A multigrid-preconditioned Newton-Krylov method for the incompressible Navier-Stokes equations, SIAM J. Sci. Comput., 23 (2001), pp. 398-418.

[86] E. Prudencio, R. Byrd, and X.-C. Cai, Parallel full space SQP Lagrange-Newton-Krylov-Schwarz algorithms for optimization problems, SIAM J. Sci. Comput., submmited.

[87] A. Quarteroni and A. Valli, Domain Decomposition Methods for Partial Differential Equations, Oxford University Press, Oxford, UK, 1999.

[88] A. Quarteroni, M. Tuveri, and Veneziani Computational vascular fluid dynamics: problems, models, and methods, Comput. Visual Sci., 2 (2000), pp. 163-197.

[89] J. N. Reddy and D. K. Gartling, The Finite Element Method in Heat Transfer and Fluid Dynamics, CRC Press, Boca Raton, FL, 2001.

[90] Y. Saad and M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsysmetric linear system, SIAM J. Sci. Stat. Comp., 7 (1986) pp. 856-869.

[91] Y. Saad, A flexable inner-outer preconditioned GMRES algorithm, SIAM J. Sci. Stat. Comp., 14 (1993), pp. 461-469.

[92] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, Philadelphia, PA, 2003.

[93] H. A. Schwarz, Gesammelte Mathematische Abhandlungen, 2 (1890), pp. 133-143. Springer Berlin. First published in Vierteljahrsschrift Naturforsch, Ges. Zürich, 26 (1870), pp. 272-286.

[94] J. N. Shadid, R. S. Tuminaro, and H. F. Walker, An inexact Newton method for fully coupled solution of the Navier-Stokes equations with heat and mass transport, J. Comput. Phys., 137 (1997), pp. 155-185.

[95] D. Silverster, H. Elman, D. Kay, and A. Wathen, Efficient preconditioning of the linearized Navier-Stokes equations, J. Comput. Appl. Math., 128 (2001), pp. 261-279.

[96] B. F. Smith, P. Bjørstad, and W. D. Gropp, Domain Decomposition: Parallel Multilevel Methods for Ellipic Partial Differential Equations, Cambridge University Press, Combridge, 1996.

[97] H. M. Tufo and P. F. Fischer, Terascale Spectral Element Algorithms and Implementations, in Proceedings of Supercomputing 99, IEEE Computer Society, 1999.

[98] R. S. Tuminaro, H. F. Walker, and J. N. Shadid, On backtracking failure in Newton-GMRES methods with a demonstration for the Navier-Stokes Equations, J. Comput. Phys., 180 (2002), pp. 549-558.

[99] R. S. Tuminaro, C. H. Tong, J. N. Shadid, K. D. Devine, and D. M. Day, On a multilevel preconditioning module for unstructured mesh Krylov solvers: Two-level Schwarz, Commum. Numer. Meth. Engrg., 18 (2002), pp. 383-389.

[100] S. Turek, A comparative study of time stepping techniques for the incompressible Navier-Stokes equations: From fully implicit nonlinear schemes to semi-implicit projection methods, Int. J. Numer. Meth. Fluids, 22 (1996), pp. 987-1011.

[101] H. A. van der Vorst, BiCGSTAB: a fast and smoothly converging variant of BiCG for the solution of nonsymmetric linear systems, SIAM J. Sci. stat. Comput., 13 (1992), pp. 631-644.

[102] F. M. White, Fluid Mechanics, McGraw-Hill, New York, NY, 1994.

[103] J. T. Wilkes, A New Methods for Solving Systems of Nonlinear Equations in Circuit Simulation, Ph.D. thesis, Duke University, Durham, NC, 1994.

[104] D. P. Young, W. P. Huffman, R. G. Melvin, C. L. Hilmes, and F. T. Johnson, Nonlinear elimination in aerodynamic analysis and design optimization, in Large-Scale PDE Constrained Optimization, L. T. Biegler et al. Eds., Springer Verlag, New York, NY, 2002, pp. 17-44.

[105] A. Zsaki, D. Rixen, and M. Paraschiviou, A substructure-based iterative inner solver coupled with Uzawa's algorithm for the Stokes problem, Int. J. Numer. Meth. Fluids, 43 (2003), pp. 215-230.

[106] CFD-ACE Thoery Manual, CFD Research Corporation, 1999.

[107] FLOW3D User Manual, AEA Industrial Technology, 1992.

[108] Spectrum$^{\mathbf{TM}}$ Theory Manual, Centric Engineering System, 1994.