# Chapter 1
# A Parallel Reduced Hessian SQP Method for Shape Optimization *

Omar Ghattas† Carlos E. Orozco‡

**Abstract**

We present a parallel reduced Hessian SQP method for smooth shape optimization of systems governed by nonlinear boundary value problems, for the case when the number of shape variables is much smaller than the number of state variables. The method avoids nonlinear resolution of the state equations at each design iteration by embedding them as equality constraints in the optimization problem. It makes use of a decomposition into nonorthogonal subspaces that exploits Jacobian and Hessian sparsity in an optimal fashion. The resulting algorithm requires the solution at each iteration of just two linear systems whose coefficients matrices are the state variable Jacobian of the state equations, i.e. the stiffness matrix, and its transpose. The construction and solution of each of these two systems is performed in parallel, as are sensitivity computations associated with the state variables. The conventional parallelism present in a parallel PDE solver—both constructing and solving the state equations in parallel—is extended to computation of the objective state variable gradient, the Lagrange multipliers, the state equation gradient with respect to each design variable, and the reduced gradient. Null space computations, as well as mesh regeneration, are performed sequentially. The central feature of this algorithm is that no additional communication routines nor parallel data structures are required to solve the shape optimization problem, beyond those needed for parallel solution of the state equations. The method has been implemented on a CM–2, and results are presented for an aerodynamic shape optimization problem, that of airfoil design in the presence of nonlinear potential flow.

## 1 Introduction

Interest has intensified in recent years in solving large-scale engineering design optimization problems. Many of these problems owe their large-scale character to their being constrained by physical laws in the form of partial differential equations (PDEs). When these laws are simulated numerically, large sparse systems of equations arise as constraints in the optimization problem. The problem of aerospace vehicle design, with its behavior described by conservation laws from such disciplines as solid mechanics, fluid flow, and heat transfer, has been the impetus for much research in this area (see, for example, [23] and the collected

*Supported by the Engineering Design Research Center, an NSF Engineering Research Center at Carnegie Mellon University. Use of the CM–2 at the Pittsburgh Supercomputing Center was made possible by grant DMC–910001P.

†Computational Mechanics Lab, Department of Civil and Environmental Engineering, and Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA 15213 (ghattas@cmu.edu).

‡Department of Civil Engineering and Applied Mechanics, University of Virginia, Charlottesville, VA 22903 (ceo7f@virginia.edu).

papers in [1] and [2]). It is clear that advanced architecture computers will be required for such problems—the simulation problem alone, which is subsumed by design optimization, has been a driving force in the push to teraflop computing. However, while much effort has been devoted to the design, analysis, and implementation of parallel algorithms for the simulation problem, little effort has been directed to the development of scalable parallel algorithms for the more difficult problem of optimal design.

In general, optimal design of systems governed by possibly nonlinear PDEs can be posed as a nonlinear optimization problem consisting of an objective function reflecting design goals, and constraints that include both the discretized form of the PDEs governing behavior (generically referred to here as the *state equations*) and any additional design constraints. Variables fall into two categories: *design variables* that describe geometry (e.g. shape, thickness), and *state variables* that describe system behavior for a fixed shape (e.g. pressure, velocity, stress, displacement, temperature). We will refer to the solution of the state equations for fixed values of the design variables as the *analysis* problem, and solution of the optimization problem as the *design* problem. Here we will focus on problems in which the design variables are parameters describing shape. In such cases, the number of state variables will be much larger than the number of design variables, and the "size" of the optimization problem can be characterized by the size of the analysis problem. For example, the problem of shape optimization of an entire aircraft, in which behavior is governed by the Euler equations of fluid flow, requires on the order of millions of state variables to describe system behavior, but on the order of hundreds or thousands of design variables to provide a sufficiently flexible parameterization of shape. While we focus on shape optimization problems, our method is applicable to any design optimization problem characterized by many state variables and far fewer design variables. In addition, we limit ourselves to problems in which objective and constraint functions are *smooth* functions of the design and state variables.

The optimization method we focus on is Sequential Quadratic Programming (SQP). Of course, there are many other techniques for nonlinearly constrained optimization problems, but the evidence is that SQP typically requires fewer iterations than competing methods (e.g. [20]). Because of the large number of state variables relative to the number of design variables, the major cost per design iteration is the cost of solving the state equations and computing gradients. Given the cost of the analysis problem, a method that tends to require the fewest total number of iterations is indeed appealing.

Here, we target problems large enough that the state equations cannot fit on a single processor; indeed, we assume the analysis problem requires the memory and speed of all processors. Such will be the case with optimal design problems of systems governed by PDEs in three spatial dimensions. We thus seek parallelism across the the dimension of the state variables. Our discussion will not be predicated on any particular parallel architecture or programming style; rather, our notion of a parallel machine is one with a "large" number of processors each possessing private memory. One important consequence is that direct methods for solving sparse linear equations are generally inefficient, due to associated data dependencies. There are many interpretations of the idea of *scalability* [21], but in our context we refer to an optimal design algorithm as having this property if an increase in the number of processors allows a concomitant increase in the size of the problem (measured in number of state variables) that can be solved efficiently.

While novel optimization techniques that map well onto parallel machines have been developed [11], their path to concurrency in based on the assumption that multiple instances of the problem can be evaluated simultaneously across processors. Specifically in optimal

design, several studies have employed parallelism, but again the approach has required that the state equations fit on each processor. In particular, parallelism has been obtained in gradient based methods by computing design sensitivities simultaneously, one sensitivity computation to a node [3],[24],[5]; alternatively, parallelism has been created in derivative-free methods by parallel evaluations of an objective function, one to a processor [35],[7]. Such approaches are not scalable to the problem sizes we consider. As the problem size (number of state variables) increases, the state equations no longer fit on a single processor, and the potential for parallelism decreases as the problem is split up into larger numbers of processors. In the limit, as the analysis problem demands the memory and speed of all processors, these algorithms reduce to their sequential counterparts.

A direct and conceptually simple route to parallelism is: eliminate the state equations (and hence state variables) at each design iteration by performing a complete analysis for a given design, making use of a parallel PDE solver. In particular, the state variables and state equations are partitioned and mapped to all processors. A consequence of the elimination of state variables is that the implicit function theorem must be used to compute gradients of the objective and remaining constraint functions (see, e.g. [17],[19]). Consequently, the derivatives with respect to each design variable require solution of a linear system of equations (the sensitivity equations) having as coefficient matrix the asymptotic Jacobian of the state equations. Thus, at least for PDE solvers that compute the Jacobian of the state equations, the same tools used for parallel solution of the state equations can be employed to solve the sensitivity equations. Having computed the state variables and state variable gradients in parallel, the remaining calculations to construct the quadratic programming subproblem (QP), and solution of the QP itself, can be performed sequentially. Under the assumption that the number of design variables is much smaller than the number of states, the linear algebraic calculations associated with solving the QP are a much smaller portion of the overall work. Thus, the parallel efficiency of such a method should be quite high for shape optimization and like problems, and implementation on a parallel machine should be straightforward, as no additional parallel tools (communication routines or data structures) are required if a parallel PDE solver is available. We refer to this method as *Nested Analysis and Design* (NAND) because the sequence of design iterates it generates entails solution of the analysis problem at each iteration. Interestingly, this method constitutes the most popular approach to solving engineering design optimization problems *sequentially* [17],[19]. If the only constraints are the state equations, this method essentially becomes the generalized reduced gradient method (GRG) [15].

While the parallel NAND method described above may be quite reasonable for analysis problems characterized by linear or weakly nonlinear boundary value problems, its requirement of completely resolving nonlinear behavior at each design iteration becomes increasingly onerous as the nonlinearity of the state equations increases. Since the state equations describing the behavior of aerospace vehicles are highly nonlinear, we are led to consider other methods that only approximately solve the state equations. In fact, by retaining the state equations as equality constraints in the optimization problem, and employing an optimization method that does not require feasibility of intermediate iterates (e.g. penalty, augmented Lagrangian, or projected Lagrangian methods [15]), the need to perform a complete analysis is obviated. The state variables are now regarded as optimization variables, and the state equations are satisfied only asymptotically, as, simultaneously, both the design variables converge to their optimal values and the state variables converge to values consistent with the optimal design. This idea of retaining state equations as equality constraints, though infrequently used, has been around in the

engineering literature for almost twenty years, typically under the name of *Simultaneous Analysis and Design* (SAND) (see the references in [30] and [18]).

While alleviating the burdensome requirement of having to resolve state equation nonlinearity at each design iteration, SAND methods introduce new difficulties. First, they require that the analysis problem be embedded in the optimization solver; thus, the use of existing analysis codes is prevented. Second, although the *dimension* of the optimization problem is unchanged—$n$ additional state variables are accompanied by $n$ additional state equations, and hence the reduced gradient vector remains the same length—the constraint Jacobian and Lagrangian Hessian matrices become very large and sparse. With a NAND method, the Jacobian and Hessian are small and dense—the largest dimension of both matrices is equal to the number of design variables, $m$. Embedding the state equations as constraints results now in Jacobian and Hessian matrices with largest dimension of $n + m$, which is much larger than $m$ in shape optimization problems. Exploiting sparsity of Jacobian and Hessian matrices in an SQP method is not a trivial task, and currently is the subject of considerable research [25]; exploiting Jacobian and Hessian sparsity *on a distributed memory parallel computer* is indeed a challenge.

In this paper we develop a parallel SAND SQP method that avoids the difficulties mentioned above, while retaining the advantage of not having to solve the state equations for each design. In particular, it does not require rewriting the analysis code into the optimization solver, and it provides optimal exploitation of Jacobian and Hessian sparsity, in the sense that storage requirements are equal to those of the NAND method. The method is based on a decomposition into nonorthogonal subspaces, and makes use of a quasi-Newton approximation to the reduced Hessian matrix. We will show that the method can be implemented so that it does not require any additional parallel data structures nor communication routines beyond those required by the parallel PDE solver. Thus, parallel implementation is as straightforward as that of the NAND method. It appears that the parallel reduced Hessian method captures the best aspects of both NAND and SAND: it combines the simple data structures and programming requirements of the parallel NAND method with the SAND idea of avoiding solution of the state equations at each iteration. In a earlier paper [28], we presented some preliminary results of an implementation of the method on the Thinking Machines CM–2. Here, we consider a modification that, on highly parallel computers, reduces the overall number of iterations, without increasing the work per iteration. In the remainder of this paper, we develop both variants of the method, describe parallel implementation in detail, and discuss results obtained from solution of a shape optimization problem in aerodynamic design on the CM–2.

For a more extensive discussion of the merits of alternative problem formulations, in particular for multidisciplinary optimization problems, the reader is directed to the recent exposition by Cramer et al. [10], who define additional formulations that are intermediate between NAND and SAND. We also note [13], in which NAND and SAND formulations are compared in the context of an optimization problem defined by one-dimensional compressible flow.

## 2 A parallel reduced Hessian SQP method

Our objective in this section is to develop an SQP method that resolves the difficulties of the large, sparse Jacobians and Hessian matrices inherent to SAND methods, and maps well onto highly parallel systems. Of course there are other ways around the size/sparsity issue—notably, an Augmented Lagrangian formulation in conjunction with a matrix-free method

for the unconstrained minimization problem, such as nonlinear conjugate gradients—but given the success of SQP (at least for small dense problems) [20], we pursue this avenue. The key to the success of the method will be its ability to exploit the structure of the state equations.

Typically, the number of design variables is much smaller than the number of state variables, and the full Hessian matrix of the Lagrangian function is sparse, indefinite, and of order of the total number of variables. It is therefore advantageous to seek a strategy that recurs a positive definite approximation to the reduced Hessian matrix, which is at least positive semidefinite at the optimum, and of order of the design variables. Such so-called *reduced Hessian* SQP methods [9],[26],[39],[4] can be very effective for problems in which the reduced Hessian is much smaller than the full Hessian matrix.

Let us begin with a canonical form of the optimization problem:

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) \\ &\text{subject to } \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ f : \Re^{n+m} \to \Re, \quad &\mathbf{h} : \Re^{n+m} \to \Re^n, \quad \mathbf{x} \in \Re^{n+m} \end{aligned} \tag{1}$$

Here, $f$ represents the objective (e.g. maximum lift/drag), and $\mathbf{h}$ is a system of nonlinear algebraic equations arising from discretization of a nonlinear boundary value problem governing the behavior of a physical system (e.g. conservation of linear momentum and mass in a viscous fluid). We have restricted constraints of the optimization problem (1) to include only the state equations. Generalization to problems with additional constraints is discussed in [27]. In the sequel, we will assume that $\mathbf{h}$ arises from a finite element approximation of a nonlinear boundary value problem (BVP), but the development goes through equally well with any other discretization scheme producing local coupling among state variables (e.g. finite volume or difference methods). The $n + m$ variables $\mathbf{x}$ consist of the $n$ state variables $\mathbf{u}$ and $m$ design variables $\mathbf{b}$. We assume that

$$n \gg m \tag{2}$$

which is usually the case for shape optimization problems, since a large number of state variables arises from discretization of the domain, but the shape is described by a relatively small number of parameters. Ultimately, the problems we wish to solve have $n = \mathcal{O}(10^{5-6})$ and $m = \mathcal{O}(10^{2-3})$.

The SQP method entails solution for a search direction $\mathbf{p}$ of a sequence of quadratic programs (QP) of the following form:

$$\text{minimize } \mathbf{p}_k^T \mathbf{g}_k + \frac{1}{2} \mathbf{p}_k^T \mathbf{W}_k \mathbf{p}_k \tag{3}$$

$$\text{subject to } \mathbf{A}_k \mathbf{p}_k = -\mathbf{h}_k \tag{4}$$

where $\mathbf{g}$ is the gradient of the objective function, $\mathbf{W}$ is the Hessian matrix of the Lagrangian function $L(\mathbf{x}, \boldsymbol{\lambda}) \equiv f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x})$, $\mathbf{A}$ is the Jacobian of the state equations, $\mathbf{p}$ is the search direction, and the subscript $k$ indicates evaluation at $\mathbf{x}_k$. For clarity we shall drop the subscript $k$; it is understood that all quantities depending on $\mathbf{x}$ are evaluated at $\mathbf{x}_k$.

To examine the structure of the constraint Jacobian $\mathbf{A}$ and Lagrangian Hessian $\mathbf{W}$, let us partition the variables into state and design components,

$$\mathbf{x}^T = [\mathbf{u}^T, \mathbf{b}^T] \tag{5}$$

in which $\mathbf{u} \in \Re^n$ and $\mathbf{b} \in \Re^m$. The partitioned constraint Jacobian takes the form:

$$\mathbf{A} = [\mathbf{A}_u \ \mathbf{A}_b] \tag{6}$$

We refer to the $n \times n$ Jacobian matrix of the state equations with respect to the state variables,

$$\mathbf{A}_u = \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \tag{7}$$

as the *stiffness matrix*, and the $n \times m$ Jacobian matrix of the state equations with respect to the design variables,

$$\mathbf{A}_b = \frac{\partial \mathbf{h}}{\partial \mathbf{b}} \tag{8}$$

as the *state equation shape Jacobian*. The stiffness matrix is typically very sparse—this reflects the average degree of the graph describing the mesh underlying the finite element approximation. Typically, this graph has low degree, since the basis functions have compact support, and are usually of low order. The Jacobian of the state equations with respect to the design variables, $\mathbf{A}_b$, can be sparse or dense, depending on the influence of the design variables. In parameter optimization (i.e. in which parameters to be optimized appear as coefficients in the PDEs), it is usually sparse, but in shape optimization, perturbation of a shape variable is typically felt by all nodes, so that $\mathbf{A}_b$ is typically dense. In any case, under the assumption (2), overall $\mathbf{A}$ is very sparse.

The structure of the Hessian $\mathbf{W}$ is similar to that of $\mathbf{A}$. Its partitioned form is

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_{uu} & \mathbf{W}_{ub} \\ \mathbf{W}_{bu} & \mathbf{W}_{bb} \end{bmatrix} \tag{9}$$

The Hessian matrix of the Lagrangian function reflects second derivatives of both the objective and state equations. The local character of finite element basis functions insures that any objective or constraint that can be expressed as a function of state variables is necessarily partially separable. In this case—and there are many physically meaningful quantities that qualify, such as stress, displacement, strain energy, velocity, pressure, lift, drag, and dissipation—the diagonal block $\mathbf{W}_{uu}$ has the same sparsity structure as $\mathbf{A}_u$. The other block matrices in $\mathbf{W}$ reflect the influence of the design variables, and, as with $\mathbf{A}_b$, can be sparse or dense. Again, their influence is too small to change the conclusion that overall $\mathbf{W}$ is very sparse. Detailed discussion of the structure of $\mathbf{W}$ and $\mathbf{A}$ for a problem in structural optimization can be found in [30].

If a solution to the QP (3) and (4) exists, it can be expressed by the solution of the Karush-Kuhn-Tucker system:

$$\begin{bmatrix} \mathbf{W} & -\mathbf{A}^T \\ -\mathbf{A} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{p} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} -\mathbf{g} \\ \mathbf{h} \end{Bmatrix} \tag{10}$$

Before we consider different methods for solving (10), let us re-emphasize that $\mathbf{p}$ is of the order of millions in the aerospace vehicle design problems we anticipate solving. A straightforward approach, when $\mathbf{W}$ is available, is to to perform a sparse factorization of the coefficient matrix of (10), as in [16]; alternatively, we may solve the Schur complement system derived from (10) iteratively, requiring the action of $\mathbf{W}^{-1}$ (which has been sparsely factored) on a vector at each iteration (i.e. a range space method [15]). Unfortunately, sparse factorization of an indefinite matrix of the large dimensions we are interested in

is not a viable option on highly parallel systems; even on a sequential machine, large fill associated with the sparsity structure of three-dimensional finite element problems would be prohibitive. Furthermore, second derivatives are rarely available in shape optimization problems. Thus, we are led to consider methods that approximate the Hessian. The major difficulty here is that standard quasi-Newton updates are inappropriate, because they generate dense matrices. One idea is to maintain an approximation of the Hessian such that a desirable sparsity pattern is achieved. Unfortunately, it is then not possible to maintain positive definiteness and satisfy the quasi-Newton condition [34]. Nevertheless, in [25] the quasi-Newton condition is dispensed with, incomplete Cholesky triangular factors of the Hessian are updated, and the resulting sparse SQP algorithm is reported to perform well. However, again, we cannot consider this approach for use on a highly parallel machine, since sparse factorization would be very inefficient due to data dependencies.

A more fruitful route to parallelization for our class of problems is to update a quasi-Newton approximation to the reduced Hessian matrix, which is of order $m$ and dense. In order to do this, let us decompose $\mathbf{p}$ into two components,

$$\mathbf{p} = \mathbf{Z}\mathbf{p}_z + \mathbf{Y}\mathbf{p}_y \tag{11}$$

in which $\mathbf{Z} \in \Re^{(n+m)\times m}$ is a matrix whose columns form a basis for the null space of $\mathbf{A}$, and $\mathbf{Y} \in \Re^{(n+m)\times n}$ is chosen so that the matrix

$$\mathbf{Q} = [\ \mathbf{Z}\ \ \mathbf{Y}\ ] \tag{12}$$

is nonsingular, and hence $\mathbf{Z}$ and $\mathbf{Y}$ form a basis for $\Re^{n+m}$. We refer to $\mathbf{p}_z \in \Re^n$ as the *null space* component, and to $\mathbf{p}_y \in \Re^m$ as the *range space* component even though strictly speaking the columns of $\mathbf{Y}$ need not span the range space of $\mathbf{A}^T$. The range space step is completely determined by substituting (11) into (4), resulting in an $n \times n$ linear system to be solved for $\mathbf{p}_y$,

$$\mathbf{A}\mathbf{Y}\mathbf{p}_y = -\mathbf{h} \tag{13}$$

where we have made use of the fact that the columns of $\mathbf{Z}$ are orthogonal to the rows of $\mathbf{A}$. The null space move is found by substituting (11) into (3) and minimizing with respect to $\mathbf{p}_z$, regarding $\mathbf{p}_y$ as fixed; the resulting optimization problem is:

$$\text{minimize } \mathbf{p}_z^T\mathbf{Z}^T\mathbf{g} + \frac{1}{2}\mathbf{p}_z^T\mathbf{Z}^T\mathbf{W}\mathbf{Z}\mathbf{p}_z + \mathbf{p}_z^T\mathbf{Z}^T\mathbf{W}\mathbf{Y}\mathbf{p}_y \tag{14}$$

The solution $\mathbf{p}_z$ to this unconstrained optimization problem is given by solving

$$\mathbf{Z}^T\mathbf{W}\mathbf{Z}\mathbf{p}_z = -\mathbf{Z}^T(\mathbf{g} + \mathbf{W}\mathbf{Y}\mathbf{p}_y) \tag{15}$$

for $\mathbf{p}_z$. The $m \times m$ reduced Hessian matrix $\mathbf{Z}^T\mathbf{W}\mathbf{Z}$ is dense but of the order of the design variables, and thus is very small compared to $\mathbf{W}$. It is therefore natural to update a quasi-Newton approximation of this matrix. A NAND method would require this storage as well, since the optimization variables in that case are just the design variables. A simple approach to handling the problematic matrix $\mathbf{Z}^T\mathbf{W}\mathbf{Y}$ is to ignore it—the resulting method has been shown to exhibit two-step Q-superlinear convergence, in the sense that

$$\frac{\| \mathbf{x}_{k+2} - \mathbf{x}^* \|}{\| \mathbf{x}_k - \mathbf{x}^* \|} \to 0 \text{ as } k \to \infty, \tag{16}$$

both using orthonormal range and null space bases [26], as well as more general bases [40].

The critical step is the definition of appropriate null and range space bases. Since $\mathbf{A}$ is large and sparse, a standard QR factorization is unacceptable, and more so on a highly parallel machine. Our guiding principle is that we would like to express $\mathbf{Z}$ and $\mathbf{Y}$ in a form that exploits the structure of $\mathbf{A}$, and that maps well onto highly parallel machines. This can be achieved by demanding that the only inverses required are those of the stiffness matrix $\mathbf{A}_u$, since this is the basic step of a Newton solver—and we know how to partition and map Newton-based PDE solvers to parallel machines. Of course we will not be computing inverses, only the action of $\mathbf{A}_u^{-1}$ on a vector. Thus, we can exploit the great strides made in parallel PDE solvers. In particular, while it is not clear how to partition and map $\mathbf{A}_b$, whose structure is problem-dependent, the structure of $\mathbf{A}_u$ derives from the geometry of the mesh, and general-purpose methods have been developed for its partitioning onto distributed processors (see, e.g. [33],[36],[12]). On the other hand, in light of the assumption (2), $\mathbf{A}_b$ is better dealt with sequentially.

The preceding criteria are satisfied with a "variable elimination" basis [15] for the null space:

$$\mathbf{Z} = \begin{bmatrix} -\mathbf{A}_u^{-1}\mathbf{A}_b \\ \mathbf{I} \end{bmatrix} \tag{17}$$

Clearly, the columns of $\mathbf{Z}$ are orthogonal to the rows of $\mathbf{A}$. Here, we write $\mathbf{A}_u^{-1}$ formally; however, we shall see that the computations can be arranged in such a way that solutions of only two linear systems involving the stiffness matrix are required, and we are free to chose the parallel method best suited to our architecture and problem (in the case of the CM–2 we end up using a diagonally-preconditioned conjugate gradient method; on a coarser grained machine, a domain decomposition method might be a better choice). A popular choice for $\mathbf{Y}$ is simply the transpose of $\mathbf{A}$ [15]—thus $\mathbf{Y}$ is orthogonal to the transpose of $\mathbf{Z}$:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{A}_u^T \\ \mathbf{A}_b^T \end{bmatrix} \tag{18}$$

Unfortunately, the linear system (13) then becomes very difficult to solve in parallel since its coefficient matrix is $\left(\mathbf{A}_u\mathbf{A}_u^T + \mathbf{A}_b\mathbf{A}_b^T\right)$. One can get around the need to form this matrix by using conjugate gradients to solve (13), and forming the required matrix-vector products on the fly. However, the resulting rate of convergence will deteriorate, since the condition number of the coefficient matrix is at least as large as the square of the stiffness matrix condition number, which already may be very large. A better approach is to dispense with the idea of orthogonal bases, and simply chose:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} \tag{19}$$

Clearly, $\mathbf{Q}$ in (12) is nonsingular provided $\mathbf{A}_u$ is nonsingular, and hence $\mathbf{Z}$ and $\mathbf{Y}$ form a basis for $\Re^{n+m}$. The invertibility of $\mathbf{A}_u$ is established by the well-posedness of the boundary value problem (provided of course that we are away from limit points). This form of basis has been used by a number of authors [14],[22],[40],[31],[29][28],[4],[32]. Substituting (19) into (13) yields the linear system to be solved for the resulting range space step, $\mathbf{p}_y$:

$$\mathbf{A}_u\mathbf{p}_y = -\mathbf{h} \tag{20}$$

We observe that this is simply a Newton step for the nonlinear system $\mathbf{h} = \mathbf{0}$. Using the null space definition (17) in (15) gives the equation to be solved for the null space move $\mathbf{p}_z$:

$$\mathbf{B}_z\mathbf{p}_z = -\mathbf{g}_z \tag{21}$$

where the reduced gradient $\mathbf{g}_z$ is defined as

$$\mathbf{g}_z \equiv \mathbf{Z}^T\mathbf{g} = -\mathbf{A}_b^T\mathbf{A}_u^{-T}\mathbf{g}_u + \mathbf{g}_b \tag{22}$$

Here $\mathbf{B}_z$ represents a quasi-Newton approximation to the reduced Hessian, and $\mathbf{g}_u$ and $\mathbf{g}_b$ represent objective gradients with respect to the state and design variables, respectively. Note that the optimality condition for the optimization problem (1)—that the reduced gradient (22) vanish—is identical to the expression for the gradient of the objective in a NAND method (in conjunction with the implicit function theorem—see [19] or [17]). This is not surprising: both are different formulations of the same problem, so both must satisfy the same optimality condition. The difference, of course, is that here, at intermediate iterations, the reduced gradient is evaluated at a value of $\mathbf{u}$ that need not satisfy $\mathbf{h} = \mathbf{0}$ for a given $\mathbf{b}$, in contrast to a NAND method, which converges the state equations for each design. Using the solutions to (20) and (21) in (11), the explicit moves in the state and design variables are:

$$\mathbf{p}_u = -\mathbf{A}_u^{-1}\mathbf{A}_b\mathbf{p}_z + \mathbf{p}_y \tag{23}$$

$$\mathbf{p}_b = \mathbf{p}_z \tag{24}$$

The expression for the update of the state variables (23) can be interpreted as being comprised of two components. The first term gives a first–order approximation of the change in state variables due to a change in design variables $\mathbf{p}_b$; the second term is the change that would occur if the shape were held constant, and the state variables were updated according to a Newton step.

The state equations are now decoupled from the optimization problem, in the following sense: all quantities related to the state equations that are needed by the SQP method can be obtained by parallel solution of linear systems involving the stiffness matrix. Just as in the NAND method, we are free to solve these systems in any fashion (and, of particular interest, on any architecture) we chose.

The steps of the method can be arranged into the following algorithm, which we refer to as Algorithm 1 (to distinguish it from a modified version to be introduced shortly):

- Initialize $k = 0$, $\mathbf{B}_z^0 = \mathbf{I}$, $\mathbf{u} = \mathbf{u}^0$, $\mathbf{b} = \mathbf{b}^0$
- Solve $(\mathbf{A}_u^0)^T\boldsymbol{\lambda}^0 = \mathbf{g}_u^0$
- $\mathbf{g}_z^0 \leftarrow -(\mathbf{A}_b^0)^T\boldsymbol{\lambda}^0 + \mathbf{g}_b^0$
- While $\|\mathbf{g}_z^k\|$ or $\|\mathbf{h}^k\| > \varepsilon$ and $k <$ `maxiter` do:
  1. Solve $\mathbf{B}_z^k\mathbf{p}_z^k = -\mathbf{g}_z$
  2. Solve $\mathbf{A}_u^k\mathbf{p}_u^k = -\mathbf{A}_b^k\mathbf{p}_z^k - \mathbf{h}^k$
  3. Find $\alpha^k$ from a line search
  4. $\mathbf{u}^{k+1} \leftarrow \mathbf{u}^k + \alpha^k\mathbf{p}_u^k$, $\mathbf{b}^{k+1} \leftarrow \mathbf{b}^k + \alpha^k\mathbf{p}_z^k$
  5. Solve $(\mathbf{A}_u^{k+1})^T\boldsymbol{\lambda}^{k+1} = \mathbf{g}_u^{k+1}$
  6. $\mathbf{g}_z^{k+1} \leftarrow -(\mathbf{A}_b^{k+1})^T\lambda^{k+1} + \mathbf{g}_b^{k+1}$
  7. $\mathbf{y}_z^k \leftarrow \mathbf{g}_z^{k+1} - \mathbf{g}_z^k$
  8. $\mathbf{B}_z^{k+1} = \mathbf{B}_z^k + \frac{1}{(\mathbf{g}_z^k)^T\mathbf{p}_z^k}\mathbf{g}_z^k(\mathbf{g}_z^k)^T + \frac{1}{\alpha_k(\mathbf{y}_z^k)^T\mathbf{p}_z^k}\mathbf{y}_z^k(\mathbf{y}_z^k)^T$
  9. $\mathbf{g}_z^k \leftarrow \mathbf{g}_z^{k+1}$

10. $k \leftarrow k+1$

- Endwhile

The vector $\boldsymbol{\lambda}$ is analogous to the adjoint variables in a NAND method, and is a first order Lagrange multiplier estimate corresponding to our choice of $\mathbf{Y}$. Note that in Step 7, $\mathbf{y}_z$ is taken as the difference of reduced gradients between two successive steps, as opposed to the reduced difference of Lagrangian gradients, i.e. $(\mathbf{Z}^k)^T(\nabla_x L(\mathbf{x}^{k+1},\boldsymbol{\lambda}^{k+1}) - \nabla_x L(\mathbf{x}^k,\boldsymbol{\lambda}^{k+1}))$. These two forms can be shown to be equivalent for the particular choice of $\mathbf{Y}$ given by (19) [27]. The update in Step 8 is the resulting BFGS formula.

Note that the primary effort per iteration of Algorithm 1 is the solution of two linear systems of equations having coefficient matrices that are transposes of each other. We refer to Step 2 as the *state variable update* and Step 5 as the *adjoint equation*. When the stiffness matrix is symmetric, as in the example of Section 4, and sparse factorization is viable (for example on a shared-memory machine with a small number of processors), the right approach would be to factorize once and perform two triangular solves. However, on highly parallel machines, these two linear systems have to be solved iteratively. For symmetric stiffness matrices, it is worth investing in a good preconditioner, since the cost of constructing it will be amortized over two solves. Furthermore, it may be desirable to apply Krylov methods appropriate for multiple righthand sides, e.g. block conjugate gradient methods. Finally, if we can afford the additional memory requirements, domain decomposition methods are a natural choice, since the subdomain factorizations can be amortized over the two linear systems solves. Of course, these benefits do not extend to the case when the stiffness matrix is unsymmetric. The advantages of these three enhancements become much more important when we consider optimization problems with additional inequality constraints beyond the equality constraints in (1); in this case, we require solution of the same coefficient matrix for $q$ righthand side vectors, where $q$ is the number of active constraints [27]. Potentially $q$ can be as large as $m$.

Tests have shown that this reduced Hessian method is more efficient than NAND even for problems that are mildly nonlinear (i.e. that require 5–7 Newton steps per iteration) [29]. However, the reduced cost of not having to solve the nonlinear state equations at each iteration is mitigated by an observed increase in the number of optimization iterations taken, relative to NAND. This is most likely due to ignoring the term $\mathbf{Z}^T\mathbf{W}\mathbf{Y}\mathbf{p}_y$ in (15). The effect of ignoring this term varies, depending on the size of $\mathbf{p}_y$, but our experience has shown an increase of typically 30% in the number of iterations, relative to NAND [27]. Biegler et al. have proposed approximating this term, either by differencing the Lagrangian gradient in the direction of $\mathbf{Y}\mathbf{p}_y$ to estimate $\mathbf{W}\mathbf{Y}\mathbf{p}_y$, or recurring a Broyden update of $\mathbf{Z}^T\mathbf{W}$ [4]. They prove that the convergence rate of the resulting method improves from two-step to one-step Q-superlinear. If we apply these approximations here, however, we must solve at each iteration an additional linear system with $\mathbf{A}_u^T$ as its coefficient matrix; since we are using iterative methods for solution, this amounts to a possible increase in work of 50% per iteration. The hope is that the reduction in optimization iterations offsets this increase in work.

We plan to examine the effectiveness of these approximations on parallel machines in the future. Here, however, we suggest a simple heuristic that has been very effective in practice. Our motivation is to modify the algorithm so that it yields exactly the same sequence of iterates as NAND for *linear* BVPs, i.e. when $\mathbf{A}_u$ is not a function of $\mathbf{u}$. This should be possible, given the form of the bases (19) and (17). Two modifications have to be made [27]: first, the righthand side of the linear system in Step 2 should be replaced by

$\mathbf{h}(\mathbf{b}^{k+1}, \mathbf{u}^k)$, and second, the stiffness matrix in that step should be evaluated at the newest value of the design variables, i.e. at $\mathbf{b}^{k+1}$. Therefore, Step 2 becomes

$$\text{Solve } \mathbf{A}_u(\mathbf{b}^{k+1}, \mathbf{u}^k)\mathbf{p}_u^k = -\mathbf{h}(\mathbf{b}^{k+1}, \mathbf{u}^k) \tag{25}$$

Note that

$$\mathbf{A}_b^k \mathbf{p}_z^k + \mathbf{h}^k \to \mathbf{h}(\mathbf{b}^{k+1}, \mathbf{u}^k) \text{ as } \mathbf{p}_z \to 0 \tag{26}$$

so that, asymptotically, the two algorithms take the same step. We refer to the method with these two modifications as Algorithm 2. In fact, Algorithm 2 is more straightforward and easier to implement than Algorithm 1, much in the same way that the Gauss Seidel stationary iterative solver is easier to implement than Jacobi. Algorithm 2 is similar, but not identical, to the Coleman-Conn reduced-SQP method [9].

For linear BVPs, Algorithm 2 takes the same number of iterations, *ipso facto*, as NAND—in fact they produce exactly the same sequence of iterates. For nonlinear boundary value problems, the two algorithms take different routes: Algorithm 2 performs a single Newton step while NAND fully converges the state equations. Our experience has been that the two are very close in the number of optimization iterations taken [29],[27].

An important distinction between Algorithms 1 and 2 arises when the stiffness matrix is symmetric. In this case, while both methods require the solution of two linear systems, the coefficient matrices of Algorithm 2 are evaluated at different points, in contrast with Algorithm 1, in which both linear systems have the same coefficient matrix. Were a direct method used to solve, this would entail an additional factorization, and Algorithm 2 would have to offer a significant reduction in iterations required to make up for this second factorization. But, for iterative solution, amortization of costs is not as extensive, so the costs should be similar. In the next section we will see how all of this can be conveniently implemented on a highly parallel system.

## 3 Parallel Implementation

In this section we discuss implementation of the parallel reduced Hessian method of the previous section on private memory parallel computers, and the CM–2 in particular. The CM–2 was chosen because it was the most powerful distributed memory machine available to us at the time this work was performed. Nevertheless, the method is not particular to fine grained SIMD machines, but is appropriate for any distributed memory parallel system, whether communication is implemented implicitly through the use of a compiler supporting global address space, or explicitly through message passing. The CM–2 processors are controlled by a microcontroller residing on a front end (FE) workstation, on which all sequential computations are performed. Therefore, when we refer to FE computations in the subsequent discussion, we imply that they are sequential portions of Algorithms 1 and 2. On MIMD systems, these computations might be carried out on a host node, or redundantly across all processors, or on a separate host machine; on a cluster of workstations, they might be conducted on a workstation designated as the master node.

To use the CM–2 effectively, we have chosen to solve the linear systems involving $\mathbf{A}_u$ in parallel, but perform all reduced space computations on the FE (since they are of dimension of the the number of design variables, which is small). Our discussion of the solution of the two "analysis-like" subproblems, the state variable update of Step 2 and the adjoint system of Step 5, will remain abstract, since the reduced Hessian method (thanks to the choice of $\mathbf{Y}$ in (19)) can leverage a wide variety of parallel solvers. In our own implementation on the CM–2, we use conjugate gradients; on an architecture with coarser granularity, it

might make more sense to use domain decomposition preconditioning, or iterative solution of the domain-decomposed Schur complement system, or even multigrid. In any case, the discussion to follow is not predicated on the use of any particular solver. We assume only that routines are available for 1) FE-to-processor broadcasting of mesh data, and 2) interprocessor communication associated with assembling element contributions across processor boundaries.

In the parallel reduced Hessian method, vectors can be divided into two types: those of dimension of the null space of $\mathbf{A}$ (i.e. of the design variables) and those of dimension of the range space (i.e. of the state variables). Therefore, we chose to treat all range space vectors as we treat the state variables, i.e. we treat them as parallel variables and they are partitioned and mapped to all processors in the same fashion as the state variables. In particular, $\mathbf{A}_{b_i}^k$ (the $i$-th column of $\mathbf{A}_b$), $\mathbf{p}_u^k$, $\boldsymbol{\lambda}^k$, and $\mathbf{g}_u^k$ share the same relationship to the underlying mesh as the state variables $\mathbf{u}^k$ and the state equations $\mathbf{h}^k$, and are therefore partitioned and mapped to the processors in the same fashion. Thus, associated with a node $j$ is the $j$-th state equation shape derivatives, the $j$-th Lagrange multiplier, the objective partial derivative with respect to the $j$-th state variable, in addition to the the $j$-th state variable and the residual of the $j$-th state equation.

We now describe the parallel implementation of Algorithm 2 in some detail.

**Step 1:** The null space move in Step 1 is computed on the FE to yield a change in the shape.

**Step 2:** A mesh generator computes, on the FE, a mesh corresponding to the new shape, which is then partitioned and mapped to processors. This is performed by the same FE-to-processor communication routine that is used to establish the initial geometry of the problem, a necessary component of any parallel PDE solver. The righthand side vector of Step 2, $\mathbf{h}(\mathbf{b}^{k+1}, \mathbf{u}^k)$, is computed elementwise, across processors, using the new mesh coordinates but old state variables. The new mesh coordinates are also used to compute the new element stiffness matrices of the coefficient matrix in Step 2, again in parallel. The linear system in Step 2 is then solved by any appropriate parallel PDE solver. In the example drawn from airfoil design in the next section, the matrix is symmetric, and we use an unstructured mesh-based, diagonally preconditioned conjugate gradient method. The stiffness matrix and righthand side vector are stored element by element; general communication is needed to assemble the elemental contributions across processor boundaries, as well as the inner products needed in each iteration of the conjugate gradient solver. Since the linear system of Step 2 is "analysis-like," the same concerns that arise in partitioning unstructured meshes for parallel PDE solvers also arise here. We use the communication precompiler from the Connection Machine Scientific Software Library (CMSSL) [38] to speed up global communication associated with assembling unstructured meshes.

**Step 3:** A line search is conducted to find a step length $\alpha$ that satisfies the strong Wolfe criteria while providing a sufficient decrease in a quadratic penalty merit function [15]. New design variables change the mesh, which is recomputed on the FE, while state equation residual recomputation is performed in parallel.

**Step 4:** The state variables are updated in parallel across all processors, while the design variables are updated sequentially on the FE.

**Step 5:** The derivative of the objective with respect to the state variables, $\mathbf{g}_u$, can be computed by differencing state variables, in parallel, if analytical derivatives are not available. Otherwise, $\mathbf{g}_u$ is computed in parallel and stored node by node. The stiffness matrix of Step 5 is then recomputed elementwise across processors to reflect the new values

of the state variables, and the adjoint system is then solved in parallel in exactly the same fashion as the state variable update in Step 2. At the end of the step, the Lagrange multipliers $\boldsymbol{\lambda}$ are stored node-by-node across all processors.

**Step 6:** In this step, $\mathbf{A}_b$ makes its first appearance. While it appears that storage of the full matrix is required, the fact that the design variables are sequential variables can be exploited to reduce the storage requirements considerably, yet result in an algorithm that exploits parallelism in the state variable dimension. Essentially the product $\mathbf{A}_b^T\boldsymbol{\lambda}$ is computed one design variable at a time. A typical calculation, for design variable $i$, proceeds as follows: the element contributions to the $i$-th column of $\mathbf{A}_b$, which is equal to $\nabla_b\mathbf{h}$, are computed in parallel at the value of the updated state and design variables. If analytical shape derivatives of the state equations are available, this is done directly, and in parallel, without FE-to-processor communication. If analytical derivatives are unavailable, a finite difference of the $i$-th design variable can be used to approximate the gradient. In this case, the shape is perturbed on the FE, the mesh is moved, and new coordinates are broadcast to processors, making use of the same FE-to-processor communication routine used in the analysis problem. The element contributions to $\mathbf{h}$ corresponding to the perturbed shape are computed in parallel, the difference is computed at the element level, and the communication precompiler is invoked to assemble the element contributions into nodal values of $\mathbf{A}_{b_i}$. The next step, the computation of the inner product $\mathbf{A}_{b_i}^T\boldsymbol{\lambda}$, is performed in parallel by multiplying nodal values of the state equation shape gradients with corresponding nodal values of the Lagrange multipliers (which are available across processors from Step 5). The subsequent sum across all nodes, needed for the inner product, is borrowed from the conjugate gradient solver for the analysis problem. The resulting scalar is added on the FE to the $i$-th component of $\mathbf{g}_b$, and the result makes up the $i$-th component of the reduced gradient $\mathbf{g}_z$, stored on the FE. A (sequential) loop over $i$ produces all components of $\mathbf{g}_z$. If this is not smaller than a prescribed tolerance $\varepsilon$, we continue to Step 7. Note that whether gradients are computed analytically or by differencing, no parallel tools are required beyond those needed for the analysis problem. Furthermore, regardless of the number of design variables, storage is needed only for a single column of $\mathbf{A}_b$.

**Steps 7–10:** These steps are carried out sequentially on the FE, in the same way as in a sequential SQP method.

The central feature of this algorithm is that it reduces the parallel shape optimization problem to a parallel analysis problem. In other words, no additional parallel tools are required beyond those needed for a parallel solver of the state equations that govern the behavior of the system being optimized. All FE-to-processor communication and interprocessor communication induced in the optimization problem can be handled by routines created for solving the state equations. Partitioning of parallel optimization variables mirrors the geometrically-based partitioning and mapping of mesh-based state variables, thus allowing us to extract parallelism from computation of the gradient of the objective with respect to the states, the Lagrange multipliers, the gradient of the state equations with respect to each design variable, and the reduced gradient.

## 4 An aerodynamic shape optimization problem

In this section we illustrate the parallel reduced Hessian SQP method by applying it to a shape optimization problem arising in aerodynamic design. The problem we consider is to find the shape of the airfoil that exhibits a desired pressure distribution. The flow around

the airfoil is modeled by the nonlinear full potential equations of compressible, inviscid flow. Though only two-dimensional, the problem captures many of the salient features of more general problems, which increase in difficulty as dimensionality, number of system parameters, nonlinearity of state equations, and coupling among the physical processes increase.

Let $\Omega$ represent the domain of definition, $\Gamma_s$ the surface of the airfoil, $\Gamma_w$ a split boundary intended to model the wake, $\Gamma_\infty$ the farfield boundary, $\mathbf{u}_\infty$ the freestream velocity, and $\alpha$ the angle of attack of the airfoil. The analysis problem is to find the pressure distribution over $\Gamma_s$. The conservation law $\nabla \cdot (\rho \mathbf{u}) = 0$ and boundary conditions take the form of the following BVP:

$$\nabla \cdot \{[1 - \frac{\gamma - 1}{2}(\nabla\phi)^2]^{\frac{1}{\gamma-1}} \nabla\phi\} = 0 \text{ in } \Omega \tag{27}$$

$$\nabla\phi \cdot \mathbf{n} = 0 \text{ on } \Gamma_s, \quad \nabla\phi \cdot \mathbf{n} = \mathbf{u}_\infty \cdot \mathbf{n} \text{ on } \Gamma_\infty \tag{28}$$

in which $\phi$ is the velocity potential, $\gamma$ is a constant and $\rho_\infty$ is the freestream density.

The shape optimization problem of finding parameters describing a shape that induces a desired pressure distribution can be stated as:

$$\text{minimize } f(\Gamma_s, \mathbf{\Phi}) = \int_{\Gamma_s} [p(\mathbf{\Phi}) - p^*]^2 \, d\Gamma \tag{29}$$

$$\text{subject to } \int_{\Omega_h} [1 - \frac{\gamma - 1}{2}(\mathbf{\Phi}^T \mathbf{Q} \mathbf{\Phi})]^{\frac{1}{\gamma-1}} \mathbf{Q} \mathbf{\Phi} \, d\Omega = \int_{\Gamma_{\infty,h}} \mathbf{N} (\rho_\infty \mathbf{u}_\infty) \cdot \mathbf{n} \, d\Gamma \tag{30}$$

where $p(\mathbf{\Phi})$ is the predicted pressure on the airfoil and $p^*$ is the desired pressure. The constraint (30) reflects the Galerkin form of the BVP, in which the finite element interpolation $\phi_h = \mathbf{N}^T \mathbf{\Phi}$ has been introduced. Here, $\mathbf{N}$ represents a vector of global basis functions and $\mathbf{\Phi}$ are nodal potentials (i.e. the state variables $\mathbf{u}$), and $\mathbf{Q} \equiv \nabla\mathbf{N} \, (\nabla\mathbf{N})^T$. The boundary conditions and additional constraints on the state variables are implied in this nonlinear algebraic system. This problem is of the form (1).

The stiffness matrix $\mathbf{A}_u$ is the Jacobian with respect to state variables of the conservation equations (30) and is given by:

$$\mathbf{A}_u(\mathbf{\Phi}) = \int_{\Omega_h} [1 - \frac{\gamma - 1}{2}(\mathbf{\Phi}^T \mathbf{Q} \mathbf{\Phi})]^{\frac{1}{\gamma-1}} \mathbf{Q} - [1 - \frac{\gamma - 1}{2}(\mathbf{\Phi}^T \mathbf{Q} \mathbf{\Phi})]^{\frac{2-\gamma}{\gamma-1}} \mathbf{Q} \mathbf{\Phi} \mathbf{\Phi}^T \mathbf{Q} \, d\Omega \tag{31}$$

This is an $n \times n$ symmetric matrix that can be constructed using standard finite element ideas. It is positive definite for subsonic flow, and indefinite for transonic flow [8].

### 4.1 Sequential performance of Algorithms 1 and 2

We first contrast the behavior of Algorithms 1 and 2 relative to NAND on a sequential machine (an IBM RS-6000 320H workstation). The topology of the mesh used is depicted in Figure 1. Although it appears to be structured, we treat the mesh as unstructured in order to test the generality of our code. The design variables are the basic parameters of the NACA family of airfoils, namely the maximum thickness $\tau$, the maximum camber $\varepsilon$, and the position of the maximum camber $\rho$. The target pressure distribution is that of a NACA 2412 airfoil at an angle of attack of one degree. The linear systems involving $\mathbf{A}_u$ are solved using MA-28, the sparse direct solver from the Harwell library. The mesh is made up 3280 finite elements and 1743 nodes. Performance of the three algorithms is tabulated in Table 1 for six different freestream Mach numbers. A typical sequence of

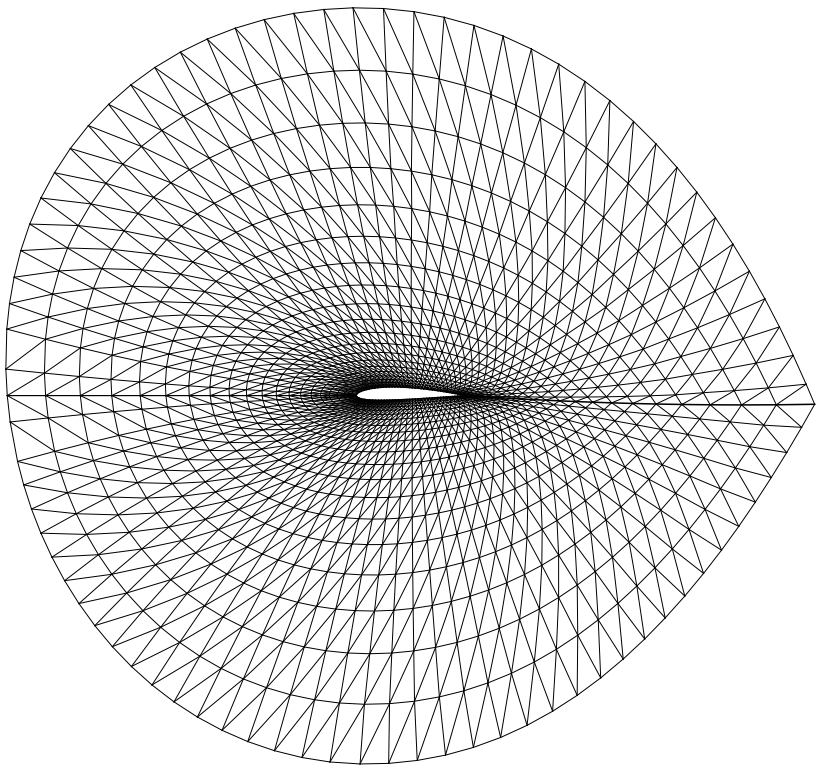

FIG. 1. *Discretized two dimensional domain with airfoil.*

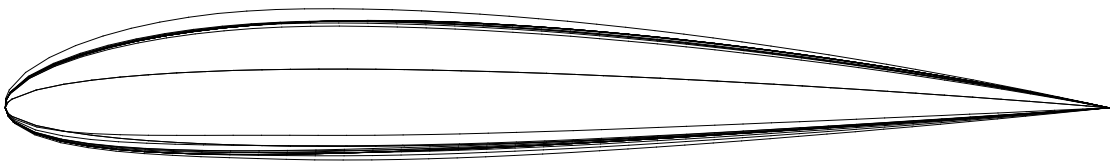

FIG. 2. *Sequence of airfoil shapes, Algorithm 1,* $M_\infty = 0.65$.

shapes is shown in Figure 2. We see that NAND requires the most CPU time, due to its need to fully solve the flow equations for each design iteration. Note the greater CPU time per optimization iteration taken by NAND for the higher Mach numbers, due to the greater degree of nonlinearity. Typically 5–7 Newton steps are required for each flow solution. The reduced Hessian methods, on the other hand, are generally unaffected by the degree of nonlinearity of the analysis problem (in fact the flow is locally supersonic for Mach 0.65, necessitating upwinding). Algorithm 1 offers a substantial reduction in CPU time, even though the number of optimization iterations increases by typically 50% over NAND. Algorithm 2 typically reduces the number of iterations, so that, with the exception of the case $M_\infty = 0.65$, the same number as NAND, or fewer, are taken. However, the need to factor two stiffness matrices per design iteration, as opposed to just one in the case of Algorithm 1, increases the work per iteration, which is reflected in relative values of CPU time per iteration in the table. In fact, this extra work per iteration offsets the modified method's reduction in the number of iterations, resulting in a slightly less efficient method—the average CPU time for Algorithm 1 is 682 seconds, while for Algorithm 2 it is 719. When an iterative method is used for solving the two systems involving $\mathbf{A}_u$, however,

TABLE 1
*Summary of sequential results, NACA 2412 airfoil.*

| NAND | | | | | | |
|---|---|---|---|---|---|---|
| $M_\infty$ | 0.40 | 0.45 | 0.50 | 0.55 | 0.60 | 0.65 |
| $f$ | 0.5440D-12 | 0.1302D-11 | 0.9653D-14 | 0.2739D-12 | 0.3576D-13 | 0.2133D-12 |
| $\| \mathbf{g}_z \|$ | 0.5043D-05 | 0.8289D-05 | 0.1065D-05 | 0.5042D-05 | 0.2227D-05 | 0.6399D-05 |
| CPU time | 964 | 975 | 1051 | 1258 | 1252 | 1385 |
| Iterations | 10 | 10 | 11 | 11 | 11 | 12 |
| Time/iter. | 96.4 | 97.5 | 95.5 | 114.4 | 113.8 | 115.4 |
| Algorithm 1 | | | | | | |
| $M_\infty$ | 0.40 | 0.45 | 0.50 | 0.55 | 0.60 | 0.65 |
| $f$ | 0.6672D-11 | 0.1642D-09 | 0.4018D-08 | 0.4230D-12 | 0.1234D-10 | 0.3074D-07 |
| $\| \mathbf{g}_z \|$ | 0.9446D-05 | 0.1216D-05 | 0.1170D-05 | 0.3890D-05 | 0.3781D-05 | 0.7041D-06 |
| CPU time | 644 | 685 | 691 | 667 | 668 | 735 |
| Iterations | 15 | 16 | 16 | 15 | 15 | 17 |
| Time/iter. | 42.9 | 42.8 | 43.2 | 44.5 | 44.5 | 43.2 |
| Algorithm 2 | | | | | | |
| $M_\infty$ | 0.40 | 0.45 | 0.50 | 0.55 | 0.60 | 0.65 |
| $f$ | 0.5063D-11 | 0.1641D-09 | 0.4018D-08 | 0.2323D-11 | 0.1223D-10 | 0.3074D-07 |
| $\| \mathbf{g}_z \|$ | 0.1016D-05 | 0.2225D-06 | 0.1343D-05 | 0.3731D-05 | 0.1106D-05 | 0.9497D-05 |
| CPU time | 637 | 639 | 639 | 699 | 703 | 997 |
| Iterations | 10 | 10 | 10 | 11 | 11 | 16 |
| Time/iter. | 63.7 | 63.9 | 63.9 | 63.5 | 63.9 | 62.3 |

our ability to exploit multiple righthand sides is not so great. Therefore, both Algorithms 1 and 2 must solve two linear systems, and hence the work per iteration is similar. Since our experience with Algorithm 2 is that it always requires fewer optimization iterations [27], we conclude that when using iterative solvers for the state variable update and the adjoint systems, Algorithm 2 will usually be more efficient than Algorithm 1. In fact, we have implemented in parallel only Algorithm 2.

## 4.2 Performance of Algorithm 2 on the CM–2

We first study the performance of Algorithm 2 on the CM–2 as the number of design variables (and hence the sequential work) increases. The problem we solve is to find the shape of the airfoil producing a pressure distribution derived from a NACA 2412 airfoil, at one degree angle of attack at a freestream Mach number of 0.6. The parameterization of the airfoil shape is depicted in Figure 3, and consists of angles and an arbitrary number

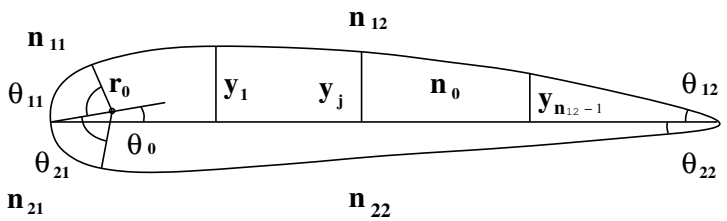


FIG. 3. *Parameterization of airfoil shape.*

of distances from the midsurface. The shape is created by fitting cubic splines through

the boundary points. As the number of design variables is increased, the flexibility of the shape parameterization increases, and we expect closer fidelity to a NACA 2412 shape. This allows us to study the behavior of our parallel shape optimization algorithm as a function of the number of design variables $m$, for fixed number of state variables, $n$. Our implementation makes use of finite difference approximations of state equation shape gradients, $\mathbf{A}_{b_i}$, and is written in CM Fortran. Table 2 illustrates the behavior of the CM–2 implementation of Algorithm 2. The results are obtained for a problem with 3952 linear

TABLE 2
*Performance on CM–2 with increasing number of design variables.*

| Opt. Vars. | 12 | 22 | 28 | 40 | 90 |
|---|---|---|---|---|---|
| $f$ | 0.113D-02 | 0.147D-03 | 0.101D-03 | 0.596D-04 | 0.278D-08 |
| $\| \mathbf{g_z} \|$ | 0.861D-03 | 0.388D-03 | 0.772D-03 | 0.955D-03 | 0.198D-03 |
| CPU time | 2566 | 3651 | 4211 | 10,718 | 29,349 |
| Iter. | 22 | 33 | 27 | 47 | 96 |
| time/iter. | 116 | 111 | 156 | 228 | 306 |

triangular elements and 2079 nodes on one sequencer (8192 processors) of the CM–2 at the Pittsburgh Supercomputing Center. The CPU time reported is total elapsed time in seconds on the FE, and reflects both parallel and sequential portions of the algorithm, as well as interprocessor communication and FE-to-processor communication. The table indicates that the increase in number of iterations roughly parallels the increase in $m$. We see that up to about 22 variables, the CPU time/iteration is insensitive to the number of design variables. As we increase $m$ further, the work per iteration increases, even though the parallel work associated with solving two systems of order 2079 remains fixed. A caveat is in order here: the elapsed time reported by the CM timer reflects other processes running on the FE. Long run times and limited resources prevented us from executing multiple runs and choosing the smallest time. Still, the trend of decreasing efficiency with increasing number of design variables (while holding the number of state variables fixed) is expected. This results from an increase both in sequential work in addition to an increase in FE-to-processor communication associated with a larger number of design variables.

The increase in sequential work (for increasing $m$ and fixed $n$) is primarily due to three factors: (1) solving a larger reduced Hessian system (Step 1) and updating a larger BFGS matrix (Step 8); (2) shape generation, since splines have to be fit to more points; and (3) the need to remesh for each additional design variable, since each column of $\mathbf{A}_b$ is computed by differencing. The first two items require only $\mathcal{O}(m^2)$ and $\mathcal{O}(m)$ work, and here, even for 90 variables, $m$ is very small (relative to conjugate gradient solution of a sparse system of size $n = 2079$). The last item, remeshing associated with each design variable, requires at least $\mathcal{O}(n)$ work, and, for the range of $m$ considered here, is certainly dominant.

FE-to-processor communication is the second source of increase of CPU time/iteration with increasing number of design variables. This is primarily due to the calculation of $\mathbf{A}_b$ by differencing. Each additional design variable requires broadcasting of new mesh coordinates from the FE host to all processors. In our implementation, we have not optimized this aspect of communication; in essence, mesh data is transferred to processors sequentially, element by element. Therefore, we should regard our reported efficiencies as providing a lower bound on expected efficiency for three reasons. First, the fact that elapsed time reflects other processes running on the FE suggests that the true efficiency is higher than computed.

Second, it is relatively straightforward to speed up the movement of mesh information from the FE to processors using CM Fortran Utility Library subroutines [37]. Third, we can do away with most of the FE-to-processor data movement, and the need to remesh $m$ times per iteration, by introducing analytical derivatives.

Finally, we present results of a scalability test to assess the behavior as $m$ and $n$ remain fixed and the number of processors is increased. The objective is unchanged, the shape is defined by 22 design variables, but 8296 nodes and 16,200 linear triangles are used to discretize the domain. We solve the problem first on 1 sequencer (8192 processors, VP ratio = 2), then on 2 sequencers (16,384 processors, VP ratio = 1). We could not afford running the problem to completion, because of its larger size in state variables (which manifests itself in a greater required number of conjugate gradient iterations). Instead, we find the time required to complete five optimization iterations. Using 8K processors, 5 iterations require 3272 seconds; 1827 seconds are needed using 16K processors. The ratio of 8K to 16K time is 0.56. If our algorithm had no sequential work, we would expect the CPU time to be cut in half, so 0.56 is quite good. Of course, this ratio will increase with more design variables. What happens to this ratio with increasing $n$ but fixed $m$ is problem-dependent: more parallel work is done (in the form of a larger number of conjugate gradient iterations for solving Steps 2 and 5) relative to the null space sequential computations, but this is tempered by the need to generate larger meshes.

## 5 Summary and ongoing work

We have presented a parallel reduced Hessian SQP method for smooth shape optimization of systems governed by nonlinear boundary value problems, for the case when the number of shape variables is much smaller than the number of state variables. The method avoids nonlinear resolution of the state equations at each design iteration by embedding them as equality constraints in the optimization problem. It makes use of a decomposition into nonorthogonal subspaces that exploits Jacobian and Hessian sparsity in an optimal fashion. The resulting algorithm requires the solution at each iteration of just two linear systems whose coefficients matrices are the state variable Jacobian of the state equations, i.e. the stiffness matrix, and its transpose. The construction and solution of each of these two systems is performed in parallel, as are sensitivity computations associated with the state variables. The conventional parallelism present in a parallel PDE solver—both constructing and solving the state equations in parallel—is extended to computation of the objective state variable gradient, the Lagrange multipliers, the state equation gradient with respect to each design variable, and the reduced gradient. Reduced space computations, as well as mesh regeneration, are performed sequentially. The central feature of this algorithm is that no additional communication routines nor parallel data structures are required to solve the shape optimization problem, beyond those needed for parallel solution of the state equations.

The algorithm has been implemented on a CM–2 and used to solve an aerodynamic shape optimization problem, that of airfoil design in the presence of nonlinear potential flow. The parallel efficiency of the algorithm, i.e. the portion of time spent on parallel tasks relative to the overall time, is high, and it scales well, for a practical number of design variables. However, the efficiency drops as the number of design variables increases, due to the associated increase in sequential computations and FE–processor communication. The parallel efficiency can be improved by incorporating analytical design variable gradients and by remeshing in parallel. Our current efforts are aimed at addressing these issues by

implementing analytical gradients and a parallel mesh moving algorithm, and replicating the sequential computations on each processor. This optimization capability is being built on top of Archimedes, an environment for solving unstructured PDEs on distributed memory parallel computers [6].

## Acknowledgments

We thank Larry Biegler for many useful discussions during the course of this work.

## References

[1] AIAA, *Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization*, 1992.

[2] ——, *Fifth AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 1994.

[3] M. AUSTIN, *Structural optimization in a distributed computing environment*, in Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, AIAA, 1992, pp. 1115–1125.

[4] L. BIEGLER, J. NOCEDAL, AND C. SCHMID, *A reduced Hessian method for large-scale constrained optimization*, SIAM Journal on Optimization, 5 (1995), pp. 314–347.

[5] C. BISCHOF, L. GREEN, K. HAIGLER, AND T. KNAUFF, *Parallel calculation of sensitivity derivatives for aircraft design using automatic differentiation*, in Fifth AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization, AIAA, 1994, pp. 73–86.

[6] G. BLELLOCH, A. FELDMANN, O. GHATTAS, J. GILBERT, G. MILLER, D. O'HALLARON, E. SCHWABE, J. SHEWCHUK, AND S.-H. TENG, *Automated parallel solution of unstructured PDE problems*, Communications of the ACM, (1995). To appear.

[7] S. BURGEE, A. GIUNTA, R. NARDUCCI, L. WATSON, B. GROSSMAN, AND R. HAFTKA, *A coarse grained variable-complexity approach to MDO for HSCT design*, in Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing, SIAM, 1995.

[8] G. CAREY AND J. ODEN, *Finite Elements: Fluid Mechanics, Vol VI*, Prentice Hall, 1986.

[9] T. COLEMAN AND A. CONN, *On the local convergence of a quasi–Newton method for the nonlinear programming problem*, SIAM Journal on Numerical Analysis, 21 (1984), pp. 755–769.

[10] E. CRAMER, J. DENNIS, P. FRANK, R. LEWIS, AND G. SHUBIN, *Problem formulation for multidisciplinary optimization*, SIAM Journal on Optimization, 4 (1994), pp. 754–776.

[11] J. DENNIS AND V. TORCZON, *Direct search methods on parallel machines*, SIAM Journal on Optimization, 1 (1991), pp. 448–474.

[12] C. FARHAT, H. SIMON, AND S. LANTERI, *TOP/DOMDEC, A software tool for mesh partitioning and parallel processing*, Computing Systems in Engineering, (1995). To appear.

[13] P. FRANK AND G. SHUBIN, *A comparison of optimization–based approaches for a model computational aerodynamics design problem*, Journal of Computational Physics, 98 (1992), pp. 74–89.

[14] D. GABAY, *Reduced Quasi-Newton methods with feasibility improvement for nonlinearly constrained optimization*, Mathematical Programming Study, 16 (1982), p. 18.

[15] P. GILL, W. MURRAY, AND M. WRIGHT, *Practical Optimization*, Academic Press, 1981.

[16] N. GOULD, *An algorithm for large-scale quadratic programming*, IMA Journal on Numerical Analysis, 11 (1991), pp. 299–324.

[17] R. HAFTKA, Z. GÜRDAL, AND M. KAMAT, *Elements of Structural Optimization*, Kluwer Academic Publishers, 1990.

[18] R. HAFTKA AND M. KAMAT, *Simultaneous nonlinear structural analysis and design*, Computational Mechanics, 4 (1989), pp. 409–416.

[19] E. HAUG AND J. ARORA, *Applied Optimal Design*, Wiley Interscience, 1979.

[20] W. HOCK AND K. SCHITTKOWSKI, *A comparative performance evaluation of 27 nonlinear programming codes*, Computing, 30 (1983), pp. 335–358.

[21] V. KUMAR AND A. GUPTA, *Analyzing scalability of parallel algorithms and architectures*, Journal of Parallel and Distributed Computing, 22 (1994), pp. 379–391.

[22] F. KUPFER AND E. SACHS, *A prospective look at SQP methods for semilinear parabolic control problems*, in Optimal Control of Partial Differential Equations, K. Hoffmann and W. Krabs, eds., Springer, 1991, pp. 145–157.

[23] P. KUTLER, *Multidisciplinary computational aerosciences*, Computing Systems in Engineering, 3 (1992), pp. 87–95.

[24] E. MALMBORG, *Design vector parallelization to speed up the structural optimization process*, in Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, AIAA, 1992, pp. 1111–1114.

[25] R. NICKEL AND J. TOLLE, *A sparse Sequential Quadratic Programming algorithm*, Journal of Optimization Theory and Applications, 60 (1989), pp. 453–473.

[26] J. NOCEDAL AND M. OVERTON, *Projected Hessian updating algorithms for nonlinearly constrained optimization*, SIAM Journal on Numerical Analysis, 22 (1985).

[27] C. OROZCO, *Large-scale shape optimization: numerical methods, parallel algorithms, and applications to aerodynamic design*, PhD thesis, Carnegie Mellon University, 1993.

[28] C. OROZCO AND O. GHATTAS, *Massively parallel aerodynamic shape optimization*, Computing Systems in Engineering, 1–4 (1992), pp. 311–320.

[29] ———, *Optimal design of systems governed by nonlinear partial differential equations*, in Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, AIAA, 1992, pp. 1126–1140.

[30] ———, *A sparse approach to simultaneous analysis and design of geometrically nonlinear structures*, AIAA Journal, 30 (1992), pp. 1877–1885.

[31] U. RINGERTZ, *Optimal design of nonlinear shell structures*, Tech. Rep. FFA TN 91-18, The Aeronautical Research Institute of Sweden, 1991.

[32] ———, *An algorithm for optimization of nonlinear shell structures*, International Journal for Numerical Methods in Engineering, 38 (1995), pp. 299–314.

[33] E. SCHWABE, G. BLELLOCH, A. FELDMANN, O. GHATTAS, J. GILBERT, G. MILLER, D. O'HALLARON, J. SHEWCHUK, AND S. TENG, *A separator-based framework for automated partitioning and mapping of parallel algorithms for numerical solution of PDEs*, in Issues and Obstacles in the Practical Implementation of Parallel Algorithms and the Use of Parallel Machines, Dartmouth, June 1992.

[34] D. SHANNO, *On variable metric methods for sparse Hessians*, Mathematics of Computation, 34 (1980), pp. 499–514.

[35] R. SHIEH, *Finite element structural analysis and optimization of space frames and trusses in a massively parallel environment*, in Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, AIAA, 1992, pp. 1069–1077.

[36] H. D. SIMON, *Partitioning of unstructured problems for parallel processing*, Computing Systems in Engineering, 2 (1991), pp. 135–148.

[37] THINKING MACHINES CORPORATION, *CM Fortran Reference Manual, Version 1.1*, 1991.

[38] ———, *CMSSL for CM Fortran, Version 3.0*, 1992.

[39] S. VASANTHARAJAN, J. VISWANATHAN, AND L. BIEGLER, *Reduced successive quadratic programming implementation for large-scale optimization problems with smaller degrees of freedom*, Computers and Chemical Engineering, 14 (1990), pp. 907–915.

[40] Y. XIE, *Reduced Hessian algorithms for solving large-scale equality constrained optimization problems*, PhD thesis, University of Colorado, Boulder, Department of Computer Science, 1991.