Computational Optimization and Applications, 4, 375–392 (1995)

Parallel Structural Optimization Applied to Bone Remodeling on Distributed Memory Machines

SHIRISH CHINCHALKAR
Advanced Computing Research Institute, Cornell Theory Center, Cornell University, Ithaca, NY 14853

THOMAS F. COLEMAN
Computer Science Department and Center for Applied Mathematics, Cornell University, Ithaca, NY 14853

Received July 23, 1993; Revised October 12, 1994

Abstract. In this paper, we investigate parallel structural optimization methods on distributed memory MIMD machines. We have restricted ourselves to the case of minimizing a multivariate non-linear function subject to bounds on the independent variables, when the objective function is expensive to evaluate as compared to the linear algebra portion of the optimization. This is the case in structural applications, when a large three-dimensional finite element mesh is used to model the structure.

This paper demonstrates how parallelism can be exploited during the function and gradient computation as well as the optimization iterations. For the finite element analysis, a 'torus wrap' skyline solver is used. The reflective Newton method, which attempts to reduce the number of iterations at the expense of more linear algebra per iteration, is compared with the more conventional active set method. All code is developed for an Intel iPSC/860, but can be ported to other distributed memory machines.

The methods developed are applied to problems in bone remodeling. In the area of biomechanics, optimization models can be used to predict changes in the distribution of material properties in bone due to the presence of an artificial implant. The model we have used minimizes a linear combination of the mass and strain energy in the entire domain subject to bounds on the densities in each finite element.

Early results show that the reflective Newton method can outperform active set methods when few variables are active at the minimum.

Keywords: structural optimization, parallel FEM

1. Introduction

Structural design applications often require the solution of large optimization problems. Designers are interested in minimizing the cost or weight of complex structures such as automobiles and bridges. Structural optimization is moving towards larger and larger problems. Not only do these problems require a significant amount of memory, they also need to be solved fast so as to speed up the overall design process and help designers study different configurations. These factors make sequential machines inadequate for solving large problems and parallelism needs to be introduced.

A popular approach for solving constrained structural optimization problems is to use an active set method with a secant update such as BFGS [8]. But as problems get larger, active set methods tend to require many iterations. Moreover, active set methods do not parallelize well on distributed memory machines. For these reasons, we suggest a new approach for solving such problems. Our approach uses the reflective Newton method [3], [4]. The Hessian is computed exactly, but is not stored explicitly. Instead, it is stored in a product form (as a sum of the product of several terms), and a preconditioned conjugate gradient

method is used to solve for the search directions. Typically, fewer iterations are required and the method can be parallelized for distributed memory machines.

In the next few sections, we described in greater detail the approach used. In section 2 we consider the problem and the motivation behind our approach. Section 3 deals with the computational techniques used in this work, and in section 4 we present the results of some numerical experiments performed in this work.

2. Problem description

Consider structural optimization problems of the following form:

$$\min_{\phi} \psi(u(\phi), \phi) \tag{1}$$
$$\text{s.t.: } L \leq \phi \leq U$$

where L and U are lower and upper bounds respectively on the vector of independent variables, ϕ. The dependent variables, $u(\phi)$, are computed by means of a finite element analysis. The most general form of structural optimization problems contains linear and non-linear equality and inequality constraints; we restrict ourselves to the simpler case here.

An area of application for the above formulation is bone remodeling. In some recent work [12], the distribution of material and physical properties of the bone is predicted using a variational model for bone reconstruction. The prediction is based on the solution of the following optimization problem:

$$\min_{\rho,t,r} \begin{Bmatrix} \mathcal{M} \\ \mathcal{U} \end{Bmatrix} \tag{2}$$

subject to:

$$L_\rho \leq \rho \leq U_\rho$$
$$L_t \leq t \leq U_t$$
$$L_r \leq r \leq U_r$$

where

$\mathcal{M}$ = mass of the structure
$\mathcal{U}$ = strain energy in the entire domain $= \frac{1}{2}\int_\Omega \underline{\sigma} \cdot \underline{\epsilon} d\Omega$
ρ = density
t = thickness
r = shape parameters
Ω = structural domain
$\underline{\sigma}$ = stress tensor
$\underline{\epsilon}$ = strain tensor
L_ρ, L_t, L_r = lower bounds on independent variables
U_ρ, U_t, U_r = upper bounds on independent variables

Solving the above optimization problem helps researchers understand the changes in the properties of human bone due to the presence of artificial implants and may ultimately lead to the design of better implants.

The Young's modulus of the bone, E, which is used in finite element analysis, is empirically related to the bone density by the relationship [12]:

$$E = C\rho^3 \tag{3}$$

where C is an empirical constant.

In this multi-objective optimization problem, a linear combination of the two objective functions is minimized. The objective function we use is $c_1\mathcal{M} + c_2\mathcal{U}$ and the coefficients c_1 and c_2 are chosen suitably. In order to solve this optimization problem numerically, the structural domain Ω is discretized into finite elements. A finite element solution gives nodal displacements, which are used to compute the strain energy, $\mathcal{U}$. The above formulation holds for a 2-D model. 3-D finite elements do not have 'thickness'; shape variables play the role of thickness in 3-D models. In [12], 2-dimensional problems were analyzed; we would like to study the applicability of the variational method of bone remodeling for 3-dimensional problems.

The applicability of the optimization problem (1) is not limited to bone remodeling. Problems arising from diverse structural applications also have similar characteristics. In structural design problems, the designer is interested in choosing model parameters (ϕ) in order to minimize an objective function such as cost or weight. Some important common characteristics of these optimization problems are:

- The objective function is a simple function of structural response, along with the input parameters (x) and the design variables (model parameters, ϕ).
- The structural response is a complex function of the design variables and the input parameters, and requires a finite element analysis.
- The input parameters are independent of the model parameters.
- The objective function, $\psi(y(\phi, x), \phi)$, is a measure of the performance of the structural system.
- Objective function evaluation is significantly more expensive than the linear algebra portion of the optimization algorithm.

Realistic finite element models of physical systems have several thousand degrees of freedom and the solution of such large problems is computationally intensive. During the optimization process, several hundred such functions may have to be evaluated. In order to speed up the optimization process parallelism can be used.

In this work, the reflective Newton method [3], [4] is used for the non-linear optimization. Among its advantages are fast convergence, ease of parallelization, and relatively few function evaluations at the expense of more linear algebra per iteration. In contrast, active set methods such as NPSOL [8] may require several hundred iterations in some cases and are also difficult to parallelize. For example, the quadratic programming subproblem in an active set method requires $O(n^2)$ work for updating a Cholesky factorization every time a variable is added or removed from the active set. It is difficult to obtain high speedups for these updates on a message passing machine. On the other hand, the reflective Newton method requires a single solution of a system of equations at every iteration and is hence easier to parallelize.

The reflective Newton method generates strictly feasible points—global and quadratic convergence results are established in [4]. The method works as follows. In iteration k a descent direction is determined. This descent direction is the first leg of a piecewise linear

A Model Interior-reflective Method

Choose $\phi^{(1)}$ in the strict interior of the feasible region.

For $k = 1, 2, \ldots$

1. Determine an initial descent direction $s^{(k)}$ for ψ at $\phi^{(k)}$. Determine a "reflective path" $p^{(k)}(\alpha)$ as described in [3].
2. Perform an approximate piecewise line minimization of $\psi(\phi^{(k)} + p^{(k)}(\alpha))$, with respect to α, to determine an acceptable stepsize $\alpha^{(k)}$.
3. $\phi^{(k+1)} = \phi^{(k)} + p^{(k)}(\alpha^{(k)})$.

Figure 1. A model interior-reflective algorithm.

"reflective path" which is searched to locate a sufficiently improved point. Details are given in [3]. A model interior-reflective method is described in Fig. 1.

How is the descent direction determined? The idea, fully described in [3], is to solve a 2-dimensional trust region subproblem. The subspace chosen, at iteration k, is defined by the scaled gradient direction and an approximate Newton step—in our case the approximate Newton step is defined by a conjugate gradient process.

The reflective Newton method is relatively new; however, initial experiments indicate that this approach generally requires relatively few (serial) major iterations to achieve good accuracy.

This research is carried out on a 32 processor Intel iPSC/860 hypercube with 8 Mbytes of local memory per processor. The processors of the iPSC/860 are connected by ethernet with peak communication speed of 2.8 Mbytes/second. Processors exchange data by sending and receiving messages. The nodes of the hypercube are attached to a front-end called the System Resource Manager (SRM) which is used to load the program on the nodes. Instead of the SRM, a Sun workstation can also be used as a front-end for the hypercube. In this work, some of the code ran on the front-end, but the computationally intensive portions ran on the hypercube. The reason for using the Sun front-end is two-fold:

- Since the active set method (NPSOL) is difficult to parallelize, it is run sequentially. However, the memory on each node of the hypercube is too small to have NPSOL running on it. Therefore, NPSOL is run on the Sun front-end.
- Proprietary code such as Matlab [9] used with the reflective Newton method is not available in source form and hence needs to be run on the Sun front-end for which a compiled executable is available.

The Portable Instrumented Communication Library (PICL) is used for message-passing [7]. Use of PICL ensures portability across several distributed memory platforms.

The next few sections describe in greater detail the approach used for solving (1). We have further restricted ourselves to problems where the independent variables are the densities in individual finite elements, i.e., we have not considered variation in the shape of the domain. During some of the discussions, we will use the objective function in Eq. (2), although the techniques are applicable with minor modifications to other objective functions as well.

3. Computational methods

We have introduced parallelism during various stages of the computation. The objective function evaluation, which includes the finite element analysis, is carried out entirely on the hypercube, along with the calculation of the gradient and the Hessian. The matrix vector products needed for the conjugate gradient method are also computed in parallel.

Two important issues in distributed memory programming are data distribution and load balancing. For structural optimization, information such as nodal coordinates and element connectivity is made available to all processors. Moreover, the torus wrap mapping which is used for storing the stiffness matrix ensures uniformity of data distribution. The element stiffness matrices are needed during the calculation of the gradient and the Hessian and are stored in a distributed fashion with each processor storing the stiffness matrices of a few elements. Load balancing among processors is also achieved easily because of the use of a torus wrap skyline factorization instead of other methods such as domain decomposition.

When NPSOL is used to solve (2), only the objective function and gradient is evaluated in parallel on the hypercube. The rest of the active set method runs on the Sun front-end.

3.1. Parallel finite element analysis

In structural finite element analysis, the following large sparse system of linear equations needs to be solved:

$$Ku = f \tag{4}$$

where K is the global stiffness matrix, u is the vector of nodal displacements and f is the vector of externally applied nodal loads. We have considered only direct methods for solving the above equation because, as detailed in the next few sections, several systems of equations with the same coefficient matrix, K, must be solved.

In this work, the matrix K is stored in a skyline form and is distributed among the processor using a torus wrap mapping [6]. In the torus wrap mapping, the processors are logically arranged in a 2-dimensional grid of size $r \times c$, where r is the number of rows in the processor grid, c is the number of columns, and $p = rc$ is the number of processors. Each processor is identified uniquely by a tuple $(i, j), i = 0..r - 1, j = 0..c - 1$. For any matrix, the entry (k, l) of the matrix is stored on processor $(k \bmod r, l \bmod c)$. In case of skyline storage, only those entries of the lower triangular portion of K that are inside the skyline are stored according to the above formula. Within each processor, the entries are stored by rows. The torus wrap mapping increases the communication bandwidth and hence reduces the communication cost. The more common row and column wrap mappings are special cases of the torus wrap mapping. However, column based methods are faster for solution of equations as the communication bandwidth is unimportant because message size is small. When the preconditioned conjugate gradient method is used with the reflective Newton method, several systems of equations need to be solved for the same factorization of K. In this case, a column based mapping for K is used. When NPSOL is used to solve (1), factorization time dominates and hence the torus wrap mapping is used with NPSOL.

Assembly of the global stiffness matrix can be performed independently by all processors; each processor is responsible for computing its portion of K. Global stiffness matrix assembly is low order work and can be done with little redundant computation.

Calculation of stresses and strains in each element can be performed in parallel without interprocessor communication. Therefore, the only message passing required is during the factorization and solution of equations and during the initial data distribution.

The parallel multifrontal method (e.g., [11] can also be used for solving the finite element system of Eq. (4). For the range of problems that could be solved on a 32 processor iPSC/860, the multifrontal method was found to be marginally faster than the skyline solver, but it required significantly more intermediate storage. Moreover, unlike the skyline solver, the storage requirements were not evenly balanced across all processors. Hence the multifrontal method could not be used to solve large problems.

3.2. *Calculation of gradient*

The gradient of the objective function with respect to the independent variable is given by the following formula:

$$\frac{\partial \psi}{\partial \phi} = \frac{\partial \tilde{\psi}}{\partial \phi} + \Lambda^T \left(\frac{\partial f}{\partial \phi} - \frac{\partial K}{\partial \phi} u \right) \tag{5}$$

where $\tilde{\psi}$ is the same as ψ, but with u and ϕ regarded as independent variables and Λ is an adjoint vector that can be computed from:

$$K\Lambda = \frac{\partial \tilde{\psi}^T}{\partial u} \tag{6}$$

Since K is already factored for solving (4), Λ can be computed by solving a system of equations given the Cholesky factor of K. If the ϕ_i's are the densities in individual finite elements, $\frac{\partial f}{\partial \phi}$ is zero. The term $\frac{\partial K}{\partial \phi} u$ is given by the following formula:

$$\frac{\partial K}{\partial \phi} u \equiv \left[\frac{\partial K}{\partial \phi_1} u \quad \frac{\partial K}{\partial \phi_2} u \cdots \frac{\partial K}{\partial \phi_{n_\phi}} u \right] \tag{7}$$

where

$$\frac{\partial K}{\partial \phi_i} u = \sum_{j=1}^{n_e} \left(\frac{\partial K_j}{\partial \phi_i} u \right) \tag{8}$$

and n_ϕ is the number of independent variables. The summation '$\sum_{j=1}^{n_e}$' corresponds to an assembly of individual element vectors ($\frac{\partial K_j}{\partial \phi_i} u$) into a global vector. When each ϕ_i is the density in exactly one finite element, $\frac{\partial K_j}{\partial \phi_i}$ is non-zero only when $i = j$ and therefore $\frac{\partial K}{\partial \phi} u$ is easy to compute. Calculation of the gradient therefore requires a solution of a system of equations with known factorization and several small matrix-vector products that can be performed independently.

3.3. *The reflective Newton method*

In this research, the reflective Newton method is used because it requires relatively few iterations (and function evaluations) for convergence and can be parallelized relatively easily. It is compared with an active set method with quasi Newton updates (NPSOL).

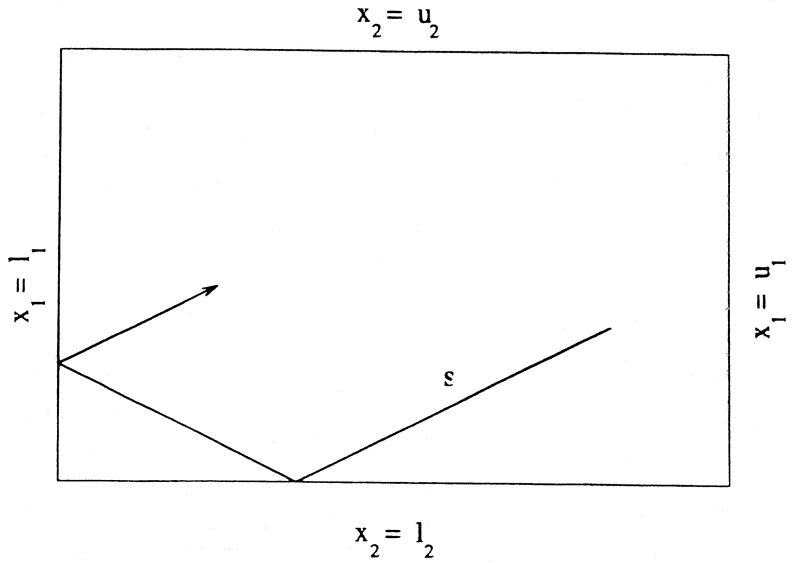


Figure 2. Line search along a reflective piecewise linear path.

Unlike NPSOL, the reflective Newton method can take advantage of an indefinite Hessian by using a direction of negative curvature. Details can be found in [3] and [4].

The reflective Newton method generates a sequence of strictly feasible iterates that ultimately converges at a quadratic rate to a local solution. The search direction, s, used in the line search is computed by solving the following trust region subproblem:

$$\min_s \left\{ s^T D_1 g + \frac{1}{2} s^T A s : \| D_1^{-1} s \|_2 \leq \Delta, s \in S \right\} \tag{9}$$

where

$$A = D_1 H D_1 + D_2 \tag{10}$$

and D_1 and D_2 are known diagonal matrices, H is the Hessian of the objective function, and g is the gradient. We limit the search space to a 2-dimensional subspace, S, defined by the scaled gradient, $D_1 g$, and either the Newton's step or a direction of negative curvature. The latter are obtained by applying the method of preconditioned conjugate gradients to

$$A s_N = b \tag{11}$$

where

$$b = -D_1 g \tag{12}$$

From a given iterate, an approximate line search is performed along a piecewise linear path. The first leg of this path is the descent direction obtained by solving (9).

Whenever a constraint is encountered, the search direction is 'reflected' against this constraint. Figure 2 shows an example of a reflective path in 2 dimensions.

The reflective Newton method requires the Hessian or an approximation to it. One of the following approaches can be used:

- A quasi Newton approximation can be used, but this ignores any negative curvature the objective function might have.

- A finite difference approximation can be computed, but it can be very expensive.
- The Hessian can be computed analytically (see section 3.4) and is cheaper than finite differences but is still expensive.
- The Hessian can be computed in a 'product form' (a sum of the product of several terms) and can be used to compute Hx for arbitrary vectors x and the preconditioned conjugate gradient method can be used to solve (11).

3.4. *Hessian of the objective function*

The Hessian is given by:

$$H \equiv [h_{ij}] \equiv \frac{\partial^2 \psi}{\partial \phi_i \partial \phi_j} = \frac{\partial^2 \tilde{\psi}}{\partial \phi_i \partial \phi_j} + \frac{\partial^2 \tilde{\psi}}{\partial \phi_i \partial u} \cdot \frac{\partial u}{\partial \phi_j} + \frac{\partial^2 \tilde{\psi}}{\partial \phi_j \partial u} \cdot \frac{\partial u}{\partial \phi_i} + \frac{\partial \tilde{\psi}}{\partial u} \cdot \frac{\partial^2 u}{\partial \phi_i \partial \phi_j} + \left(\frac{\partial u}{\partial \phi_i}\right)^T \frac{\partial^2 \tilde{\psi}}{\partial u^2} \cdot \frac{\partial u}{\partial \phi_j} \tag{13}$$

It can be shown that the Hessian is dense. Calculation of the Hessian requires calculation of $\frac{\partial u}{\partial \phi}$. This is expensive to compute because for each i, the following equation needs to be solved:

$$K \frac{\partial u}{\partial \phi_i} = -\frac{\partial K}{\partial \phi_i} u \tag{14}$$

Even though the factorization of K has been computed earlier, calculation of $\frac{\partial u}{\partial \phi}$ requires the solution of n_ϕ systems of equations (where n_ϕ is the number of independent variables). Moreover, $\frac{\partial u}{\partial \phi}$ needs to be stored as it is needed for computing various terms in the formula for the Hessian. Therefore, this approach requires a significant amount of memory.

3.5. *The preconditioned conjugate gradient method*

The preconditioned conjugate gradient method (PCG) can be used to solve (11). This approach is preferred when the Hessian is not known explicitly, but the product Hx, for any vector x, can be computed cheaply. As shown in section 3.6, this is indeed the case here. Since PCG is used only to compute a search direction, an accurate solution is not necessary. Therefore, PCG is used with a very loose convergence criterion so as to reduce the number of iterations. Solving $Ax = b$ is equivalent to minimizing $Q = \frac{1}{2}x^T Ax - b^T x$. Convergence is based on the criterion [10]:

$$\frac{k(Q_k - Q_{k-1})}{(Q_k - Q_0)} \leq \alpha \tag{15}$$

where k is the iteration number, $Q_k = \frac{1}{2}x_k^T Ax_k - b^T x_k$, x_k is the value of x at iteration k, and $x_0 = 0$. α is chosen to be 0.1. The above convergence criterion is equivalent to stopping when the reduction in Q is a small fraction of the average reduction in Q per iteration.

A diagonal preconditioner whose entries approximate the diagonal entries in H is used. Since H is not known explicitly, 3 random vectors, $x^{(1)}, x^{(2)}, x^{(3)}$, are chosen and the product $Hx^{(i)}$, $i = 1, 2, 3$ is computed. Let $[x \quad y \quad z]$ denote a matrix with the vectors

$k = 0$; $x_0 = 0$; $r_0 = b - Ax_0$; $Q_0 = 0$; $kmax = n/2$;
while $(k < kmax)$
  Solve $Mz_k = r_k$
  $k = k + 1$
  if $(k = 1)$ **then**
    $d_1 = z_0$
  else
    $\beta_k = r_{k-1}^T z_{k-1} / r_{k-2}^T z_{k-2}$
    $d_k = z_{k-1} + \beta_k d_{k-1}$
  endif
  $\gamma_k = d_k^T A d_k$
  if $(\gamma_k < 0)$ **then**
    $s_N = d_k$
    return
  endif
  $\alpha_k = r_{k-1}^T z_{k-1} / \gamma_k$
  $x_k = x_{k-1} + \alpha_k d_k$
  $r_k = r_{k-1} - \alpha_k A d_k$
  $Q_k = 0.5 x_k^T A x_k - b^T x_k$
  if $(k(Q_k - Q_{k-1})/Q_k \leq 0.1)$ **then**
    $s_N = x_k$
    return
  endif
end

Figure 3. The preconditioned conjugate gradient algorithm.

x, y, and z as its three columns. A diagonal approximation to H, viz. $\tilde{H}$, is computed by solving the following least squares problem:

$$\min \| \tilde{H}[x^{(1)} x^{(2)} x^{(3)}] - H[x^{(1)} x^{(2)} x^{(3)}] \|_F \tag{16}$$

and the preconditioner, M, is set to

$$M = D_1 \tilde{H} D_1 + D_2 \tag{17}$$

If a diagonal element M_i happens to be negative, M_i is set to 1.

The algorithm used for PCG is shown in Fig. 3. If $d_k^T A d_k$ is less than 0, the matrix A is indefinite and d_k, a direction of negative curvature, is returned.

3.6. *Hessian in product form*

As mentioned earlier, if H can be computed in a product form, and if the product Hx can be computed cheaply for any vector x, then PCG can be used to solve (11). Terms in Eq. (13) can be computed as follows:

- $\dfrac{\partial^2 \tilde{\psi}}{\partial \phi_i \partial \phi_j}$

This term is easy to compute because ψ is a simple function of ϕ. When each ϕ_i is the density in a single finite element, this matrix is diagonal.

- $\dfrac{\partial^2 \tilde{\psi}}{\partial u \partial \phi_i} \cdot \dfrac{\partial u}{\partial \phi_j}$

$$\frac{\partial^2 \tilde{\psi}}{\partial u \partial \phi_i} \cdot \frac{\partial u}{\partial \phi_j} \equiv -\frac{\partial^2 \tilde{\psi}}{\partial u \partial \phi} K^{-1} \frac{\partial K}{\partial \phi} u \tag{18}$$

$$= -\frac{\partial^2 \tilde{\psi}}{\partial u \partial \phi} K^{-1} A \tag{19}$$

The matrices A and $\frac{\partial^2 \tilde{\psi}}{\partial u \partial \phi}$ are sparse and are computed on an element-by-element basis. When forming the product $\frac{\partial^2 \tilde{\psi}}{\partial u \partial \phi} K^{-1} Ax$, calculation of $K^{-1}(Ax)$ requires the solution of one system of equations with the Cholesky factor of K computed in (4).

- $\dfrac{\partial^2 \tilde{\psi}}{\partial u \partial \phi_j} \cdot \dfrac{\partial u}{\partial \phi_i}$

This term is the transpose of the previous term. Using the objective function in Eq. (2), this term reduces to $-C_2 A^T K^{-1} A$ which is symmetric and hence is the same as the previous term. If a different objective function is used, this term would have to be computed and would also require a solution of one system of equations with known factorization of the coefficient matrix.

- $\dfrac{\partial \tilde{\psi}}{\partial u} \cdot \dfrac{\partial^2 u}{\partial \phi_i \partial \phi_j}$

Equation (4) can be differentiated twice with respect to ϕ and substituted in the above expression to obtain:

$$\frac{\partial \tilde{\psi}}{\partial u} \frac{\partial^2 u}{\partial \phi_i \partial \phi_j} = \frac{\partial \tilde{\psi}}{\partial u} K^{-1} \left[-\frac{\partial K}{\partial \phi_i} \cdot \frac{\partial u}{\partial \phi_j} - \frac{\partial K}{\partial \phi_j} \cdot \frac{\partial u}{\partial \phi_i} - \frac{\partial^2 K}{\partial \phi_i \partial \phi_j} u \right] \tag{20}$$

$$= \Lambda^T \left[-\frac{\partial K}{\partial \phi_i} \cdot \frac{\partial u}{\partial \phi_j} - \frac{\partial K}{\partial \phi_j} \cdot \frac{\partial u}{\partial \phi_i} - \frac{\partial^2 K}{\partial \phi_i \partial \phi_j} u \right] \tag{21}$$

Since $A = \frac{\partial K}{\partial \phi} u$,

$$\Lambda^T \frac{\partial K}{\partial \phi} \frac{\partial u}{\partial \phi} = \Lambda^T \frac{\partial K}{\partial \phi} K^{-1} A \tag{22}$$

When multiplying by x, the term $K^{-1}Ax$ computed earlier can be reused. Similarly, the second term is equivalent to $A^T \frac{\partial K}{\partial u} K^{-1} \Lambda$. When multiplying by x, another system of equations $K^{-1}(\Lambda x)$ needs to be solved. The last term is non-zero only if $i = j$.

- $\left(\frac{\partial u}{\partial \phi_i}\right)^T \frac{\partial^2 \tilde{\psi}}{\partial u^2} \cdot \frac{\partial u}{\partial \phi_j}$

  This can be written as $A^T K^{-1} B K^{-1} A$, where $B = \frac{\partial^2 \tilde{\psi}}{\partial u^2}$. B is computed by the following formula:

$$B = \frac{\partial^2 \tilde{\psi}}{\partial u^2} \tag{23}$$

$$= \frac{\partial^2}{\partial u^2}\left(C_1 \mathcal{M} + C_2 \sum_e 1/2 u^T K_e u\right) \tag{24}$$

$$= C_2 \sum_e K_e \tag{25}$$

  where the summation is over all bone elements. B is thus equivalent to the collection of element stiffness matrices. B is not computed explicitly; the element stiffness matrices are stored separately before assembly of K. $K^{-1}Ax$ has been computed earlier and can be reused. The above product therefore requires the solution of one system of equations, namely, $K^{-1}(BK^{-1}Ax)$.

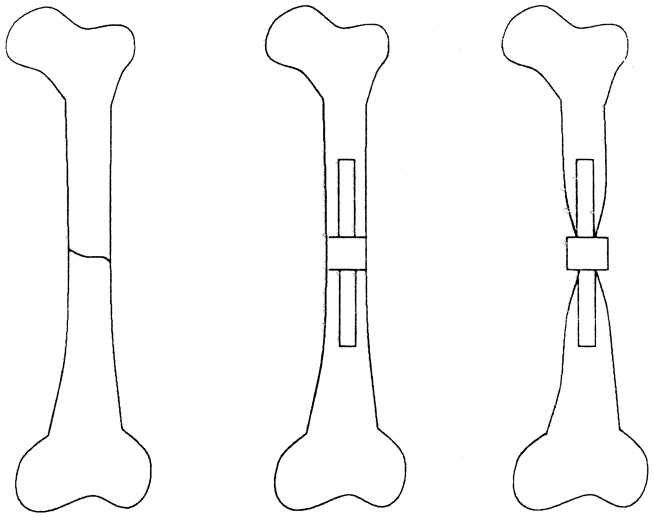
In summary, the cost of computing H in product form is negligible and the cost of computing Hx is roughly equivalent to that of solving 3 sparse systems of equations (with known factorization of the coefficient matrix).

When the above calculations are performed in parallel on the hypercube, the matrices such as $\frac{\partial K}{\partial \phi} u$ and $\frac{\partial K}{\partial \phi} \Lambda$ are computed independently in parallel and are stored and used on an element-by-element basis.

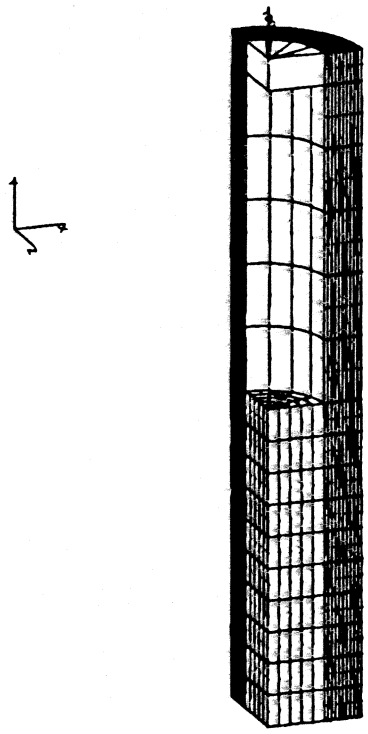
On the Intel iPSC/860, the time spent on sending a message of n bytes from a processor to its neighbor can be approximated by $75 + 0.4n$ microseconds [5]. The startup time (latency) of 75 microseconds has a noticeable effect on the overall performance when the messages are small. This is the case during forward elimination and back substitution. Since calculation of Hx requires 3 such solutions, it can be expensive. However, by solving 2 sets of equations, $K^{-1}Ax$ and $K^{-1}\Lambda x$, simultaneously, the speed of calculation of Hx can be increased.

4. Problems and results

As mentioned in an earlier section, the methods developed in this paper are applied to problems in bone remodeling. Figure 4 shows a steel stem (implant) in a fractured bone. The stiffness of steel is much greater than that of bone. As a result, the steel implant carries most of the applied load. This ultimately causes the bone to atrophy as shown in Fig. 4(a) and can result in loosening of the implant. The presence of the implant thus has an undesirable effect on the bone. It would be beneficial to see if this model can predict the actual behavior correctly. To analyze this problem, a finite element model with 844 elements is constructed (see Fig. 4(b)). In Table 1 we provide details of the problem. Different values of c_1 and c_2 are used and the optimization is carried out. The predicted density distribution in the bone is shown in Fig. 5. It can be seen that this model predicts that a significant portion of the bone would atrophy. Since the implant is very stiff and since a linear finite element analysis is performed, load transfer from the bone to the stem takes place at the top of the stem and results in negligible stresses at the bottom of the bone. If a non-linear analysis



(a) Fractured bone supported by a steel implant.



(b) Finite element mesh for bone remodeling problem in Fig. 3. Only a quarter of the physical domain is modeled.

Figure 4.

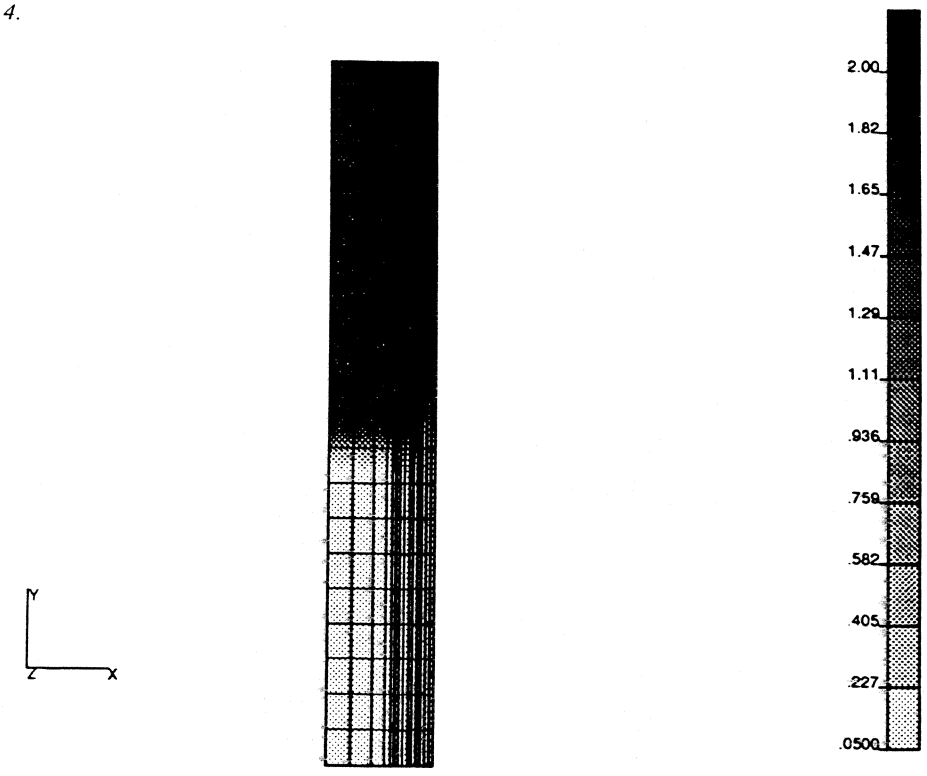


Figure 5. Optimal density distribution for the problem in Fig. 4

Table 1. Description of bone remodeling problems.

	Problem 1	Problem 2
n	600	720
n_{elems}	844	780
n_{eqns}	11496	11082
c_1	1	1
c_2	1000	25
C	2875	2875
l	0.05	0.05
u	2	2
x_{start}	1	1

Table 2. Results of NPSOL for problems in Table 1.

	Problem 1	Problem 2
# Processors	16	16
# Major iter.	5	17
# Function eval.	6	31
# Active	600	719
Real time (sec)	1087	4653
f_{opt}	2.390043×10^4	9.949413×10^3

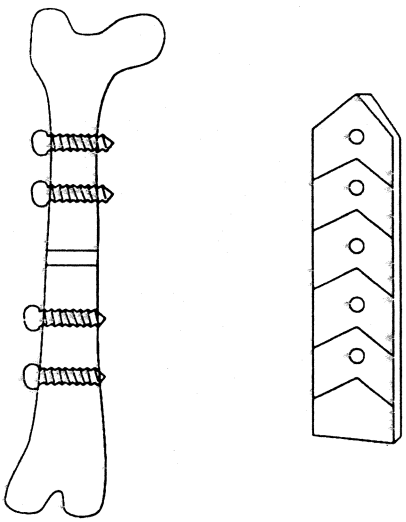


Figure 6. Canine bone and plate: the plate is attached to the bone by screws.

using interface elements had been performed, the load transfer would have taken place over a longer region and such a large portion of the bone would not have atrophied. Tables 2 and 3 show the performance of NPSOL and the reflective Newton method on the above problem. In this case, since most of the variables are at their bounds at the solution, NPSOL converges fast and the reflective Newton method takes longer.

The second example is that of a screw hole in a femur. Sometimes, a broken bone is supported by metallic plates affixed by screws (Fig. 6). After a few weeks, when the bone

Table 3. Results of reflective Newton method on problems in Table 1.

	Problem 1	Problem 2
# Processors	16	16
# Major iter	22	24
# PCG iter	50	57
# Function eval	25	26
# Hx	309	353
Real time (sec)	6459	7398
f_{opt}	2.391010×10^4	9.058229×10^3

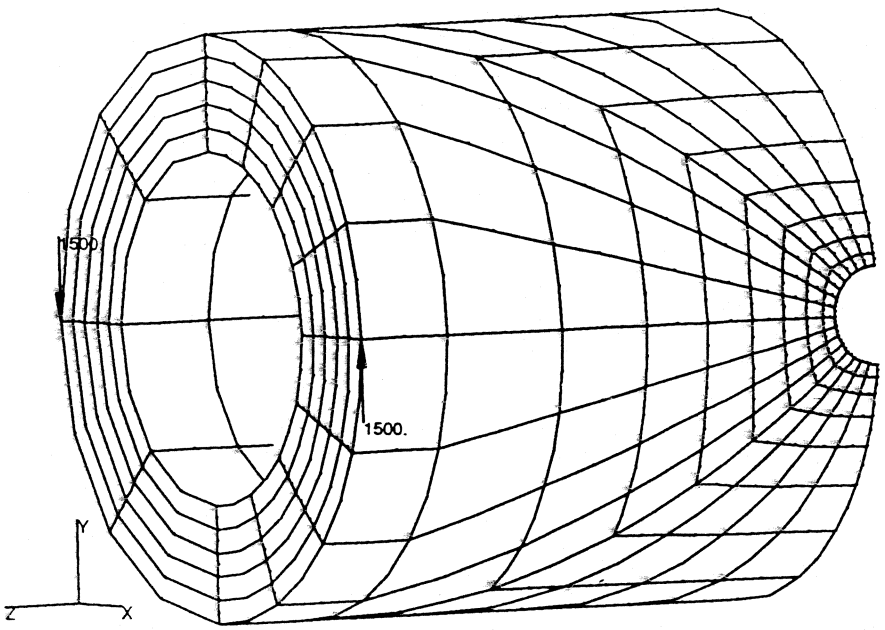


Figure 7. Finite element mesh for bone remodeling problem in Fig. 6.

heals, the plate and screws are removed. The presence of holes in the bone caused by the removal of screws results in stress concentration. Experiments performed by researchers on canine bones [1] show that after a few weeks the bone properties around the hole change in order to eliminate stress concentration and the bones become as strong as the original bones without holes. To analyze this problem, a finite element model with 780 elements is constructed Fig. 7). In Table 1 we provide details of the problem. In Tables 2 and 3 the performances of NPSOL and the reflective Newton method on this problem are detailed. Figure 8 shows the distribution of density of bone after reconstruction. The results are not entirely realistic because the model predicts some zero density areas which are not observed in practice. This may be because the shape of the domain is not allowed to vary. In experiments on canine bones, it was observed that the bone thickened near the hole. In [12], when the thickness of the 2-D model was allowed to vary, realistic results were obtained. In the 3-D case, the bone also became thinner because the stresses on the outer surface are greater than the stresses on the inner surface. However, in practice, the bone does not become thin.

Table 4. Breakup of CPU time (percent) for the reflective Newton method on problems in Table 1.

	Problem 1	Problem 2
Line search	27.4	27.2
Conjugate gradients	16.5	15.7
Function evaluation	49.7	51.4
Gradient evaluation	1.7	1.5
Other	4.7	4.2

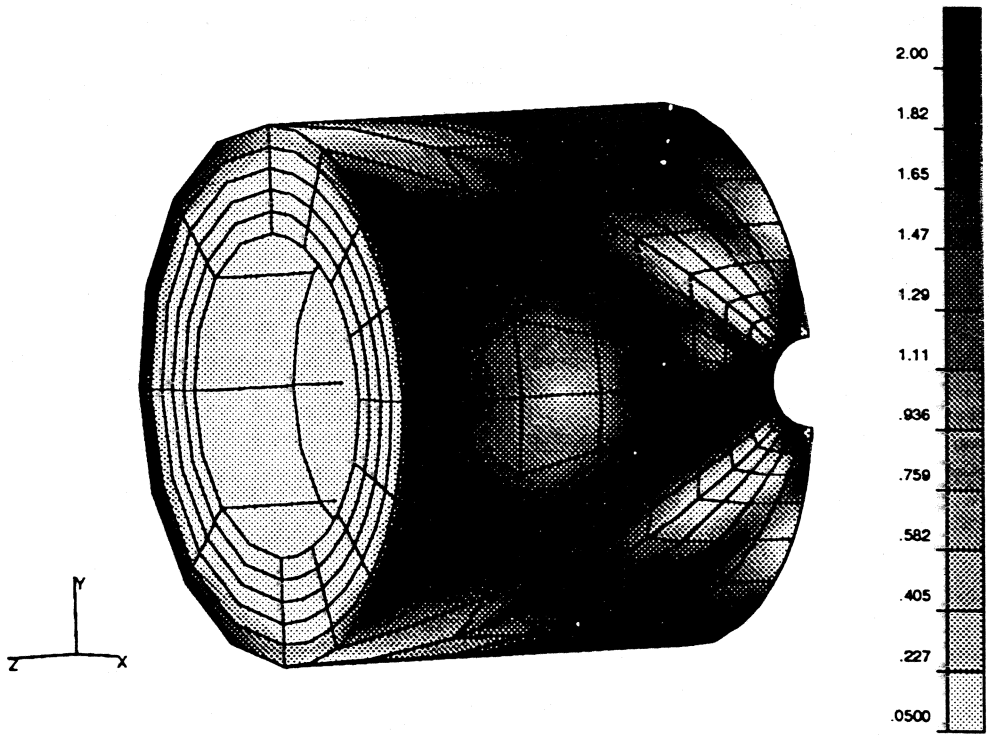


Figure 8. Optimal density distribution for the problem in Fig. 7.

In Table 4 we break down the time taken during different stages of the reflective Newton method. It can be seen that the time spent on solving for the search direction and the line search is quite significant. Function evaluations take up approximately half of the total time. However, it should be noted that the floating point speed during function evaluation is much higher than the floating point speed during PCG and line search. This is because function evaluation requires factorization, whereas PCG requires solution of a system of equations with a pre-computed factorization.

Some models are constructed to illustrate possible problems with the active set method and why the reflective Newton method might be better in these circumstances. Table 5 shows two cases. For each case, models with different numbers of elements are constructed. When few variables are active at the solution, NPSOL is found to perform poorly, requiring more than 100 iterations and more than 100 function evaluations. On the other hand, the reflective Newton method requires significantly fewer iterations and function evaluations

References

1. A.H. Burstein, J.D. Currey, V.H. Frankel, K.G. Heiple, P. Lunseth, and J.C. Vessely, "Bone strength: The effect of screw holes," Journal of Bone and Joint Surgery, Vol. 54-A, No. 6, pp. 1143–1156, 1972.
2. D.R. Carter and W.C. Hayes, "The Compressive Behavior of Bone as a Two-Phase Porous Structure," The Journal of Bone and Joint Surgery, Vol. 59-A, pp. 954–962, 1977.
3. T.F. Coleman and Y. Li, "A reflective Newton method for minimizing a quadratic function subject to bounds on some of the variables," Technical Report CTC92TR111, Cornell Theory Center, Cornell University, Ithaca, NY, 1992 (to appear in SIAM Journal on Optimization).
4. T.F. Coleman and Y. Li, "On the convergence of reflective Newton methods for large-scale nonlinear minimization subject to bounds," Mathematical Programming, Vol. 67, pp. 189–224, 1994.
5. T.H. Dunigan, "Performance of the Intel iPSC/860 and Ncube 6400 Hypercubes," ORNL/TM-11790, Oak Ridge National Laboratory, 1991.
6. R.A. van de Geijn, "Massively parallel LINPACK benchmark on the Intel Touchstone DELTA and iPSC/860 systems," Progress Report, Department of Computer Sciences, University of Texas, Austin, Texas, 1991.
7. G.A. Geist, M.T. Heath, B.W. Peyton, and P.H. Worley, "A Users' Guide to PICL: A Portable Instrumented Communication Library," ORNL/TM-11616, Oak Ridge National Laboratory, 1991.
8. P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, "User's guide for NPSOL (version 4.0): A FORTRAN package for nonlinear programming," Technical Report SOL 86-2, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1986.
9. The Math Works, Inc., PRO-MATLAB for Sun workstations, 1990.
10. S.G. Nash and A. Sofer, "Assessing a search direction within a truncated-Newton method," Operations Research Letters, Vol. 9, No. 4, pp. 219–221, 1990.
11. A. Pothen and C. Sun, "A distributed multifrontal algorithm using clique trees," Technical Report CTC91TR72, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, 1991.
12. G. Subbarayan, "Bone Construction and Reconstruction: A Variational Model and its Applications," Ph.D. Thesis, Department of Mechanical and Aerospace Engineering, Cornell University, 1990.