	$n = 120$	$n = 480$	$n = 960$	$n = 1440$
# processors	4	8	16	32
# major iter	34	59	58	41
# PCG iter	116	169	227	87
# function eval	40	68	70	41
#Hx	520	934	1026	528
Real time (sec)	1591	11032	23254	21503
f_{opt}	1.867292×10^4	1.84625×10^4	1.841288×10^4	1.662569×10^4

Table 7: Results - Reflective Newton method for problem 'hole'

	$n = 320$	$n = 640$	$n = 960$
# processors	8	16	16
# major iter	12	173	190
# function eval	15	317	362
# active	246	582	838
Real time (sec)	522	19784	38957
f_{opt}	3.83221×10^3	3.794078×10^3	3.785172×10^3

Table 8: Results - NPSOL for problem 'beam'

	$n = 320$	$n = 640$	$n = 960$
# processors	8	16	16
# major iter	35	48	60
# PCG iter	92	146	163
# function eval	39	54	66
#Hx	511	753	964
Real time (sec)	3695	10938	21715
f_{opt}	3.8179236×10^3	3.789715×10^3	3.778519×10^3

Table 9: Results - Reflective Newton method for problem 'beam'

	Problem1	Problem2
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

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

	hole	beam
n	120, 480, 960, 1440	320, 640, 960
n_{elems}	120, 480, 960, 1440	320, 640, 960
n_{eqns}	2328, 8974, 17686, 20464	6240, 11520, 16800
c_1	1	10000
c_2	1	1
C	1	1
l	0.464169	1
u	1.5	10
x_{start}	1	5

Table 5: Test problems for comparing NPSOL and reflective Newton method

	$n = 120$	$n = 480$	$n = 960$	$n = 1440$
# processors	4	8	16	32
# major iter	124	198	127	99
# function eval	270	411	256	214
# active	76	372	561	1404
Real time (sec)	2437	25281	40034	64161
f_{opt}	1.860103×10^4	1.840978×10^4	1.842017×10^4	1.668379×10^4

Table 6: Results - NPSOL for problem 'hole'

	Problem1	Problem2
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 1: Description of bone remodeling problems

	Problem1	Problem2
# 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

Table 2: Results of NPSOL for problems in table 1

	Problem1	Problem2
# 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

Table 3: Results of reflective Newton method on problems in table 1

[Subbarayan 90] Subbarayan, G., "Bone Construction and Reconstruction: A Variational Model and its Applications", Phd Thesis, Department of Mechanical and Aerospace Engineering, Cornell University, 1990.

References

[Burstein 72] Burstein, A. H., Currey, J. D., Frankel, V. H., Heiple, K. G., Lunseth, P., and Vessely, J. C., "Bone strength: The effect of screw holes", *Journal of Bone and Joint Surgery*, V54-A, N6, pp. 1143-1156, 1972.

[Carter 77] Carter, D. R. and Hayes, W. C., "The Compressive Behavior of Bone as a Two-Phase Porous Structure", *The Journal of Bone and Joint Surgery*, V59-A, pp. 954-962, 1977.

[Coleman 92a] Coleman, T. F., and Li, Y., "A reflective Newton method for minimizing a quadratic function subject to bounds on some of the variables", *Technical Report*, CTC92TR111, Nov 1992.

[Coleman 92b] Coleman, T. F., and Li, Y., "On the convergence of reflective Newton methods for large-scale nonlinear minimization subject to bounds", *Technical Report*, CTC92TR110, Nov 1992.

[Dunigan 91] Dunigan, T. H., "Performance of the Intel iPSC/860 and Ncube 6400 Hypercubes", ORNL/TM-11790, Oak Ridge National Laboratory, April 1991.

[Geijn 91] van de Geijn, R. A., "Massively parallel LINPACK be nchmark on the Intel Touchstone DELTA and iPSC/860 systems", *Progress Report*, Department of Computer Sciences, University of Texas, Austin, Texas, 1991.

[Geist 91] Geist, G. A., Heath, M. T., Peyton, B. W., and Worley, P. H., "A Users' Guide to PICL: A Portable Instrumented Communication Library", ORNL/TM-11616, Oak Ridge National Laboratory, February 1991.

[Gill 86] Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H., "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.

[MathWorks 90] The MathWorks, Inc., "PRO-MATLAB for Sun workstations", Jan 1990.

[Nash 90] Nash, S. G., and Sofer, A., "Assessing a search direction within a truncated-Newton method", *Operations Research Letters*, V9, N4, pp. 219-221, 1990.

[Pothen 91] Pothen, A. and Sun, C., "A distributed multifrontal algorithm using clique trees", *Technical Report CTC91TR72*, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, August 1991.

function evaluations. On the other hand, the reflective Newton method requires significantly fewer iterations and function evaluations (tables 6-9). From the results for the reflective Newton method, it can be seen that PCG requires approximately 3 iterations for the approximate solution of equation (9). The objective function has multiple minima for the problems solved, causing NPSOL and the reflective Newton method to give different results.

For the problem with 1440 variables, the reflective Newton method is 3 times faster than NPSOL, which suggests that for large problems, significant benefits can be realized by using the reflective Newton method. Even though we have not parallelized the entire code, the computationally intensive portion of the reflective Newton method (function evaluation, gradient evaluation, line search, and conjugate gradients) was parallelized. From table 4 it can be seen that the sequential portion of the code consumed less than 5 percent of the total time. As mentioned earlier, in case of NPSOL, because of the difficulty of parallelizing the active set method, only the objective function and gradient evaluation was parallelized. The remaining code used up approximately 10 percent of the real time. Therefore, in spite of the different extents to which parallelism is exploited in the two methods, it is apparent that for large problems, the reflective Newton method can be significantly faster than NPSOL.

5 Conclusions

This paper shows how the reflective Newton method can be used to solve structural optimization problems in parallel on distributed memory machines. The method is applied to problems in bone remodeling. For the problems solved, the bone remodeling theory did not give realistic results; further experimentation is necessary. It can also be seen that for large problems with few activities at the solution, the reflective Newton method can outperform the active set method. When the Hessian is expensive to evaluate, PCG can be used provided the product Hx is easy to evaluate.

In order to get realistic results for bone remodeling problems, several improvements are necessary. The choice of c_1 and c_2 is not obvious. For the problems in section 4, different combinations of c_1 and c_2 were tried. But in real implant design applications, it is important to be able to know the values of c_1 and c_2 that produce accurate results. It is also important to use linear and non-linear equality and inequality constraints so that this theory can be applied to other areas of structural optimization. The problem of multiple minima needs to be addressed as it is not computationally feasible to explore the entire design space and it is not easy to attribute any meaning to local minima. It might also be beneficial to incorporate shape variables in the analysis as the shape of the bone can change during reconstruction.

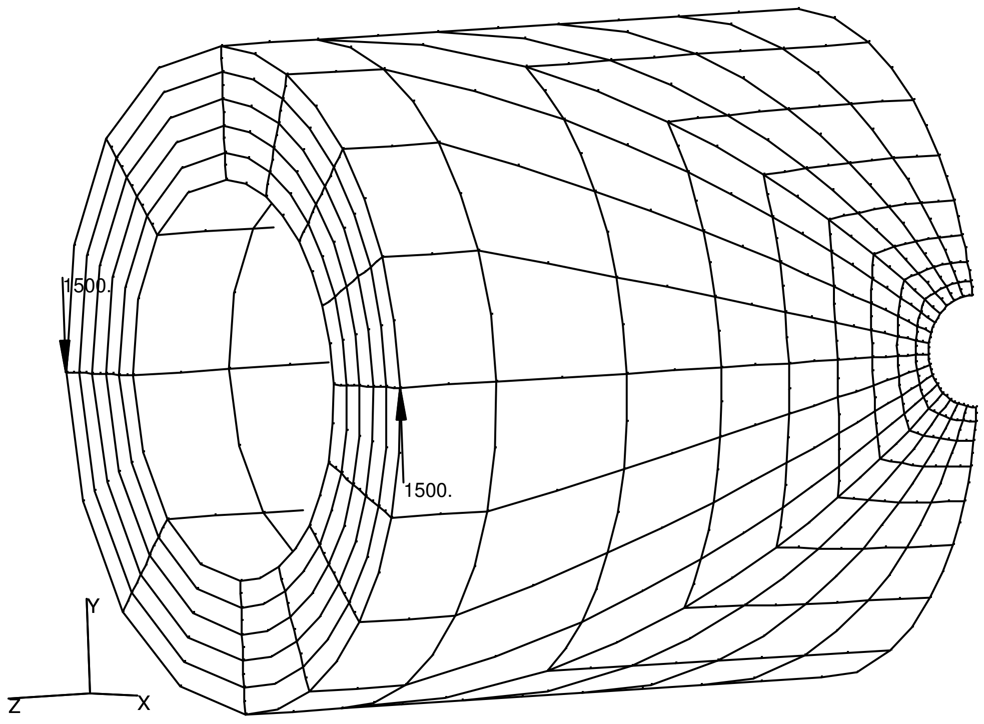

Figure 6: Finite element mesh for bone remodeling problem in figure 5

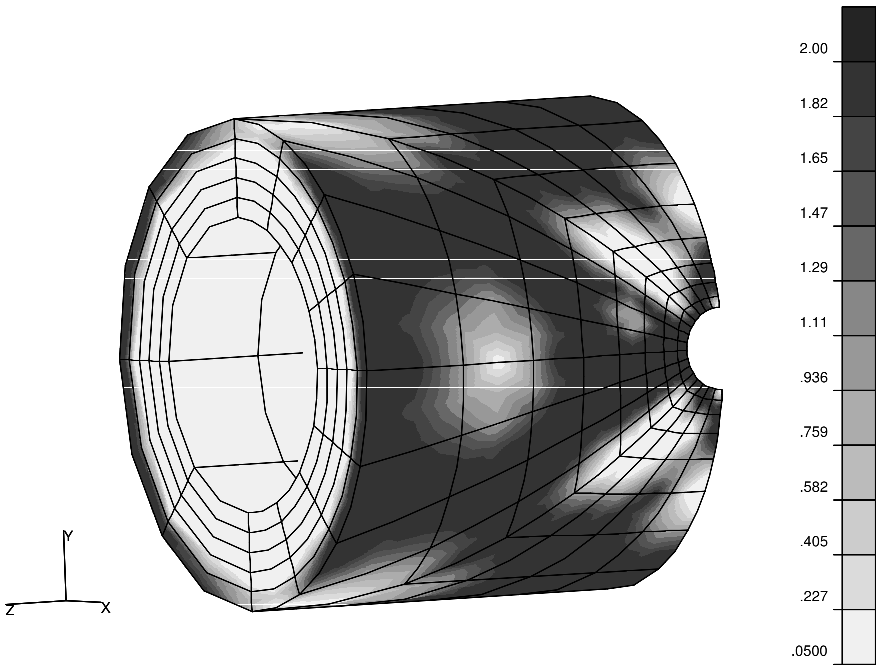

Figure 7: Optimal density distribution for the problem in figure 6

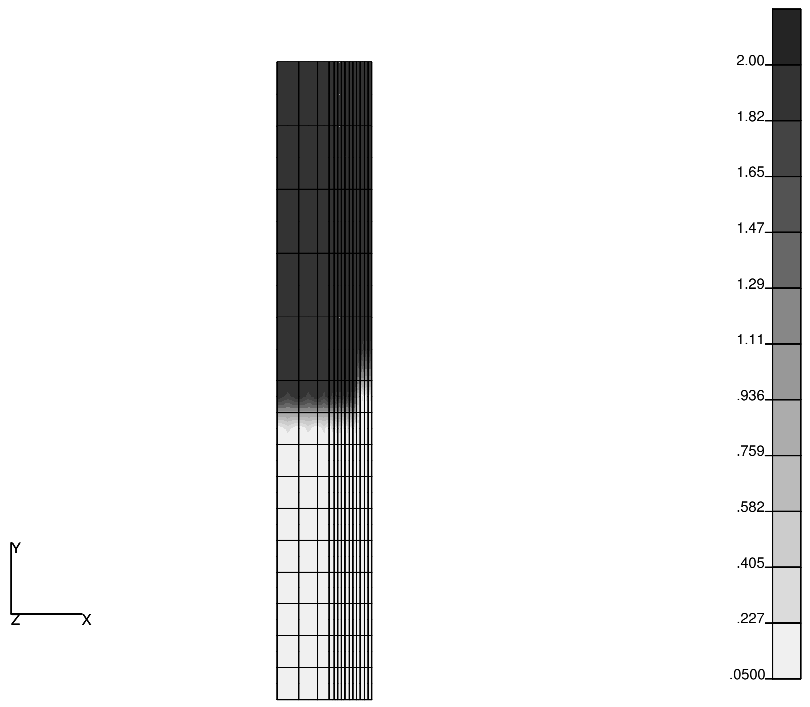

Figure 4: Optimal density distribution for the problem in figure 3

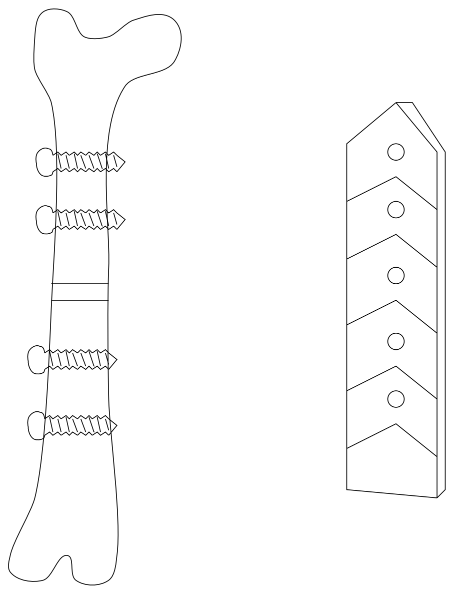

Figure 5: Canine bone and plate: the plate is attached to the bone by screws

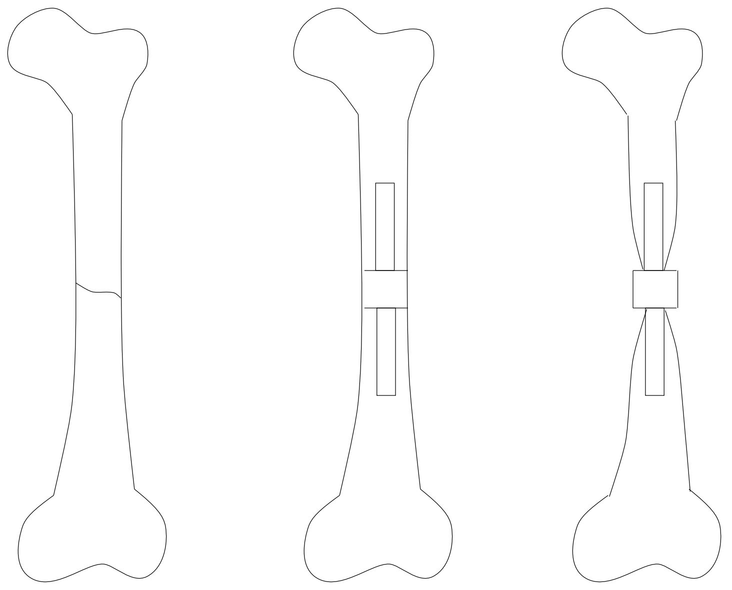

Figure 2: Fractured bone supported by a steel implant

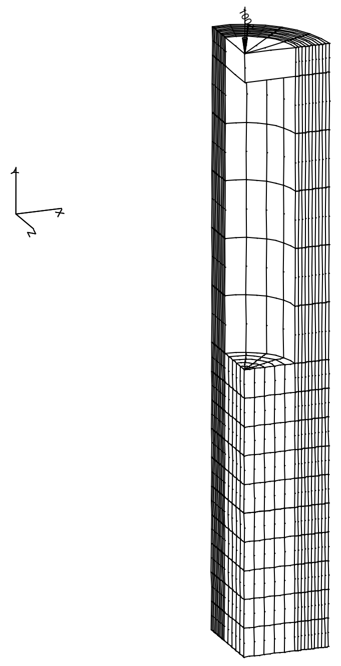

Figure 3: Finite element mesh for bone remodeling problem in figure 2. Only a quarter of the physical domain is modeled

out. Figure 4 shows the predicted density distribution in the bone. 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 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. Because 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 (figure 5). After a few weeks, when the bone 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 ([Burstein 72]) 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 (figure 6). Table 1 shows some details of the problem. Tables 2 and 3 show the performance of NPSOL and the reflective Newton method on this problem. Figure 7 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 [Subbarayan 90], 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. But in practice, the bone does not become thin. Unless the mass of the marrow is included in the analysis, this effect will be observed. Using different values for the ratio c_1/c_2, different density distributions will be obtained. However, it is not clear as to how that ratio should be chosen.

Table 4 shows the breakup of 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.

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

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

  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:

$$\begin{aligned} B &= \frac{\partial^2 \tilde{\psi}}{\partial u^2} && (22) \\ &= \frac{\partial^2}{\partial u^2}\left(C_1 \mathcal{M} + C_2 \sum_e 1/2 u^T K_e u\right) && (23) \\ &= C_2 \sum_e K_e && (24) \end{aligned}$$

  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 ([Dunigan 91]). 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 2 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 2 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 figure 3). Table 1 shows some details of the problem. Different values of c_1 and c_2 are used and the optimization is carried

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

- $\frac{\partial^2 \tilde{\psi}}{\partial u \partial \phi_i} . \frac{\partial u}{\partial \phi_j}$

$$\begin{aligned} \frac{\partial^2 \tilde{\psi}}{\partial u \partial \phi_i} . \frac{\partial u}{\partial \phi_j} &\equiv -\frac{\partial^2 \tilde{\psi}}{\partial u \partial \phi} K^{-1} \frac{\partial K}{\partial \phi} u && (17) \\ &= -\frac{\partial^2 \tilde{\psi}}{\partial u \partial \phi} K^{-1} A && (18) \end{aligned}$$

  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 $\frac{\partial^2 \tilde{\psi}}{\partial u \partial \phi} K^{-1} A x$, calculation of $K^{-1}(Ax)$ requires the solution of one system of equations with the Cholesky factor of K computed in (4).

- $\frac{\partial^2 \tilde{\psi}}{\partial u \partial \phi_j} . \frac{\partial u}{\partial \phi_i}$
  This term is the transpose of the previous term. Using the objective function in equation (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.

- $\frac{\partial \tilde{\psi}}{\partial u} . \frac{\partial^2 u}{\partial \phi_i \partial \phi_j}$
  Equation (4) can be differentiated twice with respect to ϕ and substituted in the above expression to obtain:

$$\begin{aligned} \frac{\partial \tilde{\psi}}{\partial u} \frac{\partial^2 u}{\partial \phi_i \partial \phi_j} &= \frac{\partial \tilde{\psi}}{\partial u} K^{-1} \left[-\frac{\partial K}{\partial \phi_i} . \frac{\partial u}{\partial \phi_j} - \frac{\partial K}{\partial \phi_j} . \frac{\partial u}{\partial \phi_i} - \frac{\partial^2 K}{\partial \phi_i \partial \phi_j} u \right] && (19) \\ &= \Lambda^T \left[-\frac{\partial K}{\partial \phi_i} . \frac{\partial u}{\partial \phi_j} - \frac{\partial K}{\partial \phi_j} . \frac{\partial u}{\partial \phi_i} - \frac{\partial^2 K}{\partial \phi_i \partial \phi_j} u \right] && (20) \end{aligned}$$

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

$$\Lambda^T \frac{\partial K}{\partial \phi} \frac{\partial u}{\partial \phi} = \Lambda^T \frac{\partial K}{\partial \phi} K^{-1} A \qquad (21)$$

  When multiplying by x, the term $K^{-1} A x$ 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 $i = j$.

```
$k = 0$; $x_0 = 0$; $r_0 = b - Ax_0$; $Q_0 = 0$; $kmax = n/2$;
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
                $s = d_k$
                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
                $s = x_k$
                return
        endif
end
```

Figure 1: The preconditioned conjugate gradient algorithm

and the preconditioner, M, is set to

$$M = D_1 \tilde{H} D_1 + D_2 \tag{16}$$

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

Figure 1 shows the algorithm used for PCG. 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 (9). Terms in equation (12) can be computed as follows:

3.4 Hessian of the objective function

The Hessian is given by:

$$H \equiv [h_{ij}] \equiv \frac{\partial^2 \psi}{\partial \phi_i \partial \phi_j} = \frac{\partial^2 \tilde{\psi}}{\partial \phi_i \partial \phi_j} + \frac{\partial^2 \tilde{\psi}}{\partial \phi_i \partial u} \cdot \frac{\partial u}{\partial \phi_j} + \frac{\partial^2 \tilde{\psi}}{\partial \phi_j \partial u} \cdot \frac{\partial u}{\partial \phi_i} + \frac{\partial \tilde{\psi}}{\partial u} \cdot \frac{\partial^2 u}{\partial \phi_i \partial \phi_j} + \left(\frac{\partial u}{\partial \phi_i}\right)^T \frac{\partial^2 \tilde{\psi}}{\partial u^2} \cdot \frac{\partial u}{\partial \phi_j} \tag{12}$$

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:

$$K \frac{\partial u}{\partial \phi_i} = -\frac{\partial K}{\partial \phi_i} u \tag{13}$$

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 (9). This approach is preferred if the Hessian is not known explicitly, but the product Hx, for any vector x, can be computed cheaply. As shown in section 3.6, it is indeed the case. 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 ([Nash 90]):

$$\frac{k(Q_k - Q_{k-1})}{(Q_k - Q_0)} \leq \alpha \tag{14}$$

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, are chosen and the product Hx is computed. A diagonal approximation to H, viz. $\tilde{H}$ is computed by solving the following least squares problem:

$$\min \|\tilde{H}x - Hx\|_F \tag{15}$$

and the summation '$\sum_{j=1}^{n_e}$' corresponds to an assembly of individual element vectors $\left(\frac{\partial K_j}{\partial \phi_i} u\right)$ 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 the active set method with quasi Newton updates (NPSOL). 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 [Coleman 92a] and [Coleman 92b].

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

$$As = b \tag{9}$$

where

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

$$b = -D_1 g \tag{11}$$

where D_1 and D_2 are known diagonal matrices, H is the Hessian of the objective function, and g is the gradient. If A is positive definite, s is a direction of descent. If A is indefinite, a direction of negative curvature needs to be computed from the above equation.

The reflective Newton method therefore 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 (9).

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 (2), 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., [Pothen 91]) can also be used for solving the finite element system of equations (4). It is marginally faster than the skyline solver, but it requires significantly more memory. Hence it cannot be used to solve large problems.

3.2 Calculation of gradient

The gradient of the objective function with respect to the independent variables 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 equivalent to ψ, but with u and ϕ regarded as independent variables and Λ is an adjoint vector that can be computed from:

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

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.

$$\frac{\partial K}{\partial \phi} u \equiv \left[\begin{array}{cccccc} \frac{\partial K}{\partial \phi_1} u & \frac{\partial K}{\partial \phi_2} u & . & . & . & \frac{\partial K}{\partial \phi_{n_\phi}} u \end{array} \right] \tag{7}$$

where

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

The next few sections describe in greater detail the approach used for solving (2). 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 equation (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 shown in the next few sections, one needs to solve several systems of equations with the same coefficient matrix, K, which makes iterative methods unattractive.

In this work, the matrix K is stored in a skyline form and is distributed among the processors using a torus wrap mapping ([Geijn 91]). 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

- 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.
- $\psi(y(\phi, x), \phi)$ 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 must be introduced.

In this work, the reflective Newton method ([Coleman 92a], [Coleman 92b]) 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 ([Gill 86]) 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.

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 ([MathWorks 90]) 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 ([Geist 91]). Use of PICL ensures portability across several distributed memory platforms.

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 = $\frac{1}{2}\int_\Omega \underline{\sigma}.\underline{\epsilon} d\Omega$
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 problem would help researchers understand the changes in the properties of human bone due to the presence of artificial implants and can 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 ([Carter 77]):

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

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 [Subbarayan 90], 2-dimensional problems were analyzed; we would like to study the applicability of the variational method of bone remodeling for 3-dimensional problems.

It is important to note that 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).

1 Introduction

Structural design applications often require the solution of a large optimization problem. 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 towards solving structural optimization problems is to use an active set method with a BFGS update. But as problems get larger, active set methods tend to require hundreds of 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 ([Coleman 92a], [Coleman 92b]). 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 direction. Typically, fewer iterations are required and the method can be parallelized on distributed memory machines.

The next few sections describe in greater detail the approach used. Section 2 describes the problem and the motivation behind solving this problem. Section 3 describes the computational techniques used in this work, and section 4 describes the results of some numerical experiments performed in this work.

2 Problem description

Consider structural optimization problems of the following form:

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

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 ([Subbarayan 90]), the distribution of material and physical properties of the bone was predicted using a variational model for bone reconstruction. The prediction was based on the solution of the following optimization problem:

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

Parallel Structural Optimization Applied to Bone Remodeling on Distributed Memory Machines[*]

Shirish Chinchalkar[†] and Thomas F. Coleman[‡]

Abstract

This paper demonstrates parallel structural optimization methods on distributed memory MIMD machines. We have restricted ourselves to the simpler 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-wrapped' 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 it 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 a significant number of variables are not active at the minimum.

Keywords: structural optimization, parallel finite element analysis

[*]Partially supported by NSF New Technologies Program, # 9108787, and by the Cornell Theory Center, which receives major funding from the National Science Foundation and IBM Corporation, with additional support from the State of New York and its Corporate Research Institutes.

[†]Advanced Computing Research Institute, Cornell Theory Center, Cornell University, Ithaca, NY 14853.

[‡]Computer Science Department, Cornell University, Ithaca, NY 14853.