1 Introduction

Normal Coordinate Analysis (NCA) (Wilson et al. 1955) plays an important role in the study of vibrational and thermal properties of various molecular structures at the atomic level. In particular, molecular vibrations at low temperature can be characterized by a collection of fundamental (normal) vibrational modes. These modes correspond to eigenvectors of a matrix that describes the atomic force interaction within the molecular system. The NCA information makes it much easier to link theoretical models to experimentally accessible macroscopic data, thus enabling a straightforward interpretation of molecular events in the absence of classical chaos. The immediate applications of NCA include characterizing thermal stability of polymer materials (Fukui et al. 200) and assessing the dynamic role protein vibration plays in the photosynthetic center of green plants (Renger 1998).

The long term benefits include the development of new materials that can be used for nano-manufacturing and bioengineering (Gupta et al. 1992).

Until recently, the use of NCA has been limited to relatively small molecules. This limitation can be partly attributed to the lack of efficient numerical algorithms for performing NCA and the memory-size limitations of traditional sequential and shared-memory parallel computers. In Noid et al. (2000), we presented a numerical scheme that is well suited for molecular systems of modest size; this scheme was successfully used to compute the lowest 100 vibrational modes for a 6000-atom (18000 degrees of freedom) polyethylene particle. This computation was performed on a single processor of NEC-SX4, a vector machine with 2 giga flops (Gflops) peak performance and 8 giga bytes of shared main memory. Although the performance achieved on the SX-4 is quite satisfactory, the large amount of memory required to carry out this calculation makes it difficult to extend the current computational approach to larger molecular systems. It was evident that in order to solve problems that are 10–100 times larger, either the algorithm would have to be modified to dramatically reduce memory usage, or it would have to be parallelized to take advantage of the

---

**References**

larger amount of memory available on distributed-memory parallel computers. Because the distributed memory programming model does not provide a global name space, parallel algorithms are implemented using a local address space on each processor with explicit inter-processor communication through message passing.

In this paper, we pursue the concept of carrying out large-scale NCA on distributed-memory parallel computer systems including multiprocessors and networks of workstations (NOWS). Computer architectures of this type are becoming increasingly popular in the scientific computing community because of their low cost/performance ratio and the portability of software made possible by the MPI standard for message passing. Our work indicates that scalable performance for large-scale NCA calculations can be achieved using distributed memory machines if a carefully designed parallelization scheme is used.

The paper is organized as follows. In the next section, we present the mathematical model of our computational task. Numerical methods and parallelization strategies are discussed in Section 3. In particular, we will review the implicitly restarted Lanczos algorithm and the shift-invert technique for extracting low frequency vibrational modes. The shift-invert technique requires us to perform a sparse LDLᵀ or Cholesky matrix decomposition followed by a sequence of sparse triangular substitutions. The latter become a bottleneck on distributed memory machines because of the relatively large latency of inter-processor communication. In Section 4, we discuss the role of a “selective inversion” technique to replace inefficient parallel substitution by latency-tolerant matrix-vector multiplication. In Section 5, we provide empirical results on the performance of our parallel NCA application on a number of polyethylene particles and crystals.

2 Mathematical model

The dynamics of a molecular system consisting of \( n \) atoms can be described by

\[
\frac{d}{dt}\left(\frac{\partial T}{\partial q_i}\right) + \left(\frac{\partial V}{\partial q_i}\right) = 0,
\]

where \( T \) and \( V \) are total kinetic and potential energy of the system, and \( q_i \) is the mass adjusted displacement of an atom in one of the Cartesian coordinate directions. If we assume molecules vibrate near an equilibrium configuration, the equation of motion can be simplified to yield

\[
\ddot{q} + F(q) = 0,
\]

where \( F \) is obtained by taking the second derivative of the potential with respect to the Cartesian coordinates, i.e.,

\[
F_{i,j} = \frac{\partial^2 V}{\partial q_i \partial q_j}.
\]

The standard technique for solving this second order ordinary differential equation (ODE) is to first diagonalize \( F \) by computing its eigenvalues and eigenvectors. This allows us to decouple the system and solve each equation independently. Each eigenvector of \( F \) is called a normal (vibrational) mode and the corresponding eigenvalue is proportional to the square of the vibrational frequency associated with that particular mode. This procedure of decomposing molecular vibration into a number of linearly independent modes is called Normal Coordinate Analysis (NCA). It is well known that the near-equilibrium motion can often be captured by a linear combination of several low frequency modes. Thus, our primary interest is to compute a number of smallest eigenvalues and corresponding eigenvectors of \( F \).

In the rest of this paper, we will consider performing NCA on computer-generated polyethylene (PE) particle and crystal models of various sizes. An \( m \)-atom model involves \( n = 3m \) degrees of freedom and is thus associated with a force interaction matrix \( F \) of dimension \( 3m \). The particles are labeled as \( pp3k \), \( pp6k \), \( pp12k \) and \( pp24k \) where the trailing number represents the number of atoms in the particle. The crystals are labeled using \( pc \) followed by the number of atoms. The geometry of each type of molecule is distinctive. A 6000-atom particle model (\( pp6k \)) is shown in Figure 1. It is generated by colliding, annealing and rotating a sequence of randomly coiled polymer chains with a chain length of 100 beads (Barnes et al., 1999; Kung et al., 1998); a single bead contains a combination of \( CH_2 \) and \( CH_3 \) groups. This process ultimately yields a spherically shaped particle that resembles a PE droplet that can be generated experimentally (Barnes et al., 1999). Figure 2 shows a 6000-atom orthorhombic PE crystal constructed by placing 60 chains of 100 atoms each in appropriate lattice positions. The potential field of PE particles is well understood and interactions between atoms are categorized as either bonded or non-bonded. These interactions are often measured in terms of various angles and internal stretches and are subsequently used to generate the force interaction matrix as described below.

The bonded interaction contributed by the \( i \)th atom can be expressed by (Boyd 1968)

\[
V^i_b(r_{i,i+1},\theta_{i,i+1,i+2},\tau_{i,i+1,i+2,i+3}) = \frac{1}{2} k_i (r_{i,i+1} - r_{0,i+1})^2
\]
scheme is presented in Tuzun et al. (1996) to speed up the calculation. The details on efficient derivative calculation is beyond the scope of this paper. We refer readers to Tuzun et al. (1996, 1997) for more information. The force interaction matrix $F$ is both symmetric and sparse. The symmetry arises from the observation that the second derivative of our potential is independent of the order in which it is taken. The sparsity results from neglecting the long range non-bonded interactions.

3 Computing normal modes in parallel

In this section we describe the key algorithmic components required for normal coordinate analysis. Implementations of many of these algorithms are available in the public domain and are based on earlier work by several researchers (Raghawan 1998, 1999, Maschhoff and Sorensen 1996). Our discussion is concerned with the issues of efficiency and scalability when these component parts are combined to compute the normal modes on parallel computers.

We begin with an overview of “implicitly restarted Lanczos,” the primary algorithm for computing a small number of extremal eigenpairs of the sparse matrix $F$. We next discuss how to accelerate the convergence of this method by means of a “shift-invert” process. The latter in turn requires the solution of sparse linear systems where the coefficient matrix is $F - \sigma I$ and $\sigma$ is a target shift near the eigenvalues of interest. Furthermore, linear systems of this form have to be solved for a sequence of right hand side vectors. These linear solutions comprise a large fraction of the total time and hence we need efficient latency-tolerant schemes for this subproblem. Effective parallelization also requires data mapping that minimizes communication requirements between the various components.
3.1 IMPLICITLY RESTARTED LANCZOS (IRL)

As described earlier in Section 2, potential models for large-scale molecular systems often neglect the interaction between atoms that are far apart. Consequently, F is typically sparse, and the cost of performing matrix-vector multiplication is low. Since we are mainly interested in a small number of eigenvalues and eigenvectors of F, the “implicitly restarted Lanczos” (IRL) (Sorensen 1992) appears to be an appropriate method for computing the desired normal modes.

To compute k eigenpairs, the IRL algorithm begins with k + p steps of the standard Lanczos iteration which produces an orthonormal basis \( \{v_1, v_2, \ldots, v_{k+p}\} \) of a Krylov subspace \( \{v_0, Fv_0, \ldots, F^{k+p-1}v_0\} \), where \( v_i = v_0 / \|v_0\| \). If we let \( V_{k+p} = (v_1, v_2, \ldots, v_{k+p}) \) and \( T_{k+p} = V_{k+p}^T F V_{k+p} \), then it is well known that

\[
FV_{k+p} = V_{k+p} T_{k+p} + f_{k+p} e_{k+p}^T,
\]

where \( V_{k+p}^T f_{k+p} = 0 \). Equation (1) will be called a Lanczos factorization (Lehoucq et al. 1999, p. 49) throughout this paper. The matrix \( T_{k+p} \) is tridiagonal. It represents the projection of \( F \) into the subspace spanned by columns of \( V_{k+p} \). Eigenvalue and eigenvector approximations can be extracted from \( \text{span}(V_{k+p}) \) through a standard Rayleigh-Ritz procedure (Parlett 1980). If the approximate eigenpairs are not sufficiently accurate, we modify (1) by applying p orthogonal transformations of size \( k+p \) by \( k+p \) to (1) from the right. If we use \( Q_{k+p} \) to denote the product of these transformations, this modification can be described by

\[
F(V_{k+p} Q_{k+p}) = (V_{k+p} Q_{k+p}) (Q_{k+p}^T T_{k+p} Q_{k+p}) + f_{k+p} e_{k+p}^T Q_{k+p}.
\]

The orthogonal transformations accumulated in \( Q_{k+p} \) are constructed such that

\[
V_{k+p} Q_{k+p} e_{i+1} = \gamma (F - \mu_1 I)(F - \mu_2 I) \ldots (F - \mu_p I) v_i,
\]

where \( \gamma \) is a normalization constant. By placing the shifts \( \mu_1, \mu_2, \ldots, \mu_p \) at appropriate locations, we can effectively filter out the unwanted spectral components from the first column of \( V_{k+p} Q_{k+p} \).

Due to the tridiagonal structure of each orthogonal transformation, the leading k columns of (2) can be shown to satisfy a new Lanczos factorization (implicitly) restarted from the vector \( V_{k+p} Q_{k+p} e_{i+1} \). We can extend this new factorization to complete an IRL cycle by performing \( p \) additional Lanczos steps. The orthogonality of the Lanczos vectors, \( v_j, j = k + 1, \ldots, k + p \), are strictly enforced by the Daniel, Gragg, Kaufman, and Stewart (DGKS) scheme (Daniel et al. 1976). As the unwanted spectral components are filtered out from the starting vector of the IRL run, accurate approximations to the desired eigenpairs begin to emerge.

The implicit restarting mechanism allows a \( k + p \) dimensional Krylov subspace to be updated using \( p \) (instead of \( k + p \)) matrix vector multiplications. By keeping \( p \) relatively small, the orthogonality of the columns of \( V_{k+p} \) can be easily maintained, and the issue of spurious eigenvalues (Parlett 1980) can be avoided.

The implicit restarting technique provides a means to accelerate the convergence of the standard Lanczos iteration by repeatedly modifying the starting vector. However, the convergence rate of the Lanczos process also depends on the relative gap between the desired eigenvalues. It has been observed that the smallest eigenvalues of \( F \) are typically clustered (Noid et al. 2000). The tiny relative gaps between the small eigenvalues could lead to slow convergence of IRL. In Figure 3, we plot the residual history of the first and the seventh approximate eigenpairs. The dimension of the Krylov subspace constructed in the IRL process is \( k + p = 100 \). For this particular run, we requested \( k = 50 \) eigenpairs. The convergence tolerance is set to \( \text{tol} = 10^{-6} \). Thus, the number of matrix vector multiplications used in each IRL cycle is roughly 50. At the end of 300 IRL cycles, only 20 eigenpairs converged. The residual curves show some irregular bumps during the course of convergence. This phenomenon indicates IRL detects nearby eigenpairs first before it converges to the desired one.

3.2 ACCELERATING THE CONVERGENCE OF IRL

A commonly used technique for accelerating the Lanczos process is to apply the Lanczos iteration to the shifted and inverted matrix \( (F - \sigma I)^{-1} \), where \( \sigma \) is a target shift near the eigenvalues of interest. Under the mapping \( \psi(\lambda) = 1/(\lambda - \sigma) \), clustered eigenvalues of \( F \) near the shift \( \sigma \) become dominant and well-separated eigenvalues of \( (F - \sigma I)^{-1} \). Therefore, they can be easily captured by the Lanczos process. This strategy is often known as “shift-invert”.

If we use \( V_j \) to denote a matrix consisting of the leading \( j \) columns of \( V_{k+p} \) and \( v_j \) the \( j \)th column of \( V_{k+p} \), then the \( j \)th step of a shift-invert Lanczos iteration involves the following computation

1. \( w \leftarrow (F - \sigma I)^{-1} v_j \);
2. \( w \leftarrow w - V_j (V_j^T w) \);
3. \( v_{j+1} \leftarrow w / \|w\| \).

The first step of this computation requires decomposing \( F - \sigma I \) into \( LDL^T \), where \( L \) is lower triangular and \( D \) is diagonal. If we are interested in a small number (for
example, less than a hundred) of normal modes, this decomposition needs to be done only once and the triangular factor $L$ and the diagonal matrix $D$ are reused in all subsequent Lanczos iterations to provide solutions to linear systems of the form:

$$(F - \sigma I)w = v_j, \quad j = 1, \ldots, k + p.$$ 

If a few low frequency normal modes are of interest, we can place $\sigma$ slightly to the left of the origin. The signs of the diagonal elements of $D$ can provide diagnostic information on potential errors introduced in the force matrix calculation. If $F$ is computed correctly, all diagonal entries of $D$ should be positive. In this case, the $LDL^T$ decomposition can actually be replaced by a Cholesky factorization $F - \sigma I = LL^T$.

If a large number of normal modes are needed, it may be advantageous to combine several “shift-invert” IRL runs, each associated with a different target shift, to compute subsets of the desired normal modes successively. Because $F - \sigma I$ becomes indefinite when $\sigma$ is larger than the smallest eigenvalue of $F$, it may be necessary to apply appropriate pivoting strategies in the $LDL^T$ factorization to maintain numerical stability. Although it is possible to utilize the techniques proposed in Duff and Reid (1983), Jones and Patrick (1994), and Grimes et al. (1998) without a significant change in data distribution, introducing partial pivoting in an $LDL^T$ factorization tends to increase the complexity of the algorithm. The development of an efficient and reliable parallel $LDL^T$ factorization procedure is currently an active research area. In addition to pivoting, we may also perform iterative refinements during a triangular substitution to improve solution accuracy whenever necessary (Li and Demmel 1999).

Although matrices encountered in our molecular models are fairly sparse, the cost of matrix decomposition and triangular solution constitutes a significant portion of the entire computation. The decomposition $F - \sigma I = LDL^T$ also requires more memory to hold the triangular factor $L$ because $L$ typically contains more nonzero entries than those in $F$. Thus the efficiency of the entire NCA calculation depends largely on the computational speed of sparse matrix decomposition and triangular substitutions.

Most serial sparse factorization codes use either a supernodal or a multifrontal (Ng and Peyton 1993, Duff and van der Vorst 1990, Rothberg and Gupta 1993) approach. In our earlier NCA calculations using a supernodal code similar to the one presented in Ng and Peyton (1993), we found that the primary limitation is the growth in memory requirements with matrix size (Noid et al. 2000). However, the acceleration in convergence is substantial.

---

Fig. 3 The relative residual history of the 1st and 7th approximate eigenpairs of $F$ computed by IRL.
Table 1: The CPU time and memory required to compute the lowest 100 vibrational modes of PE particles of various sizes

<table>
<thead>
<tr>
<th>n</th>
<th>CPU time (seconds)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9000</td>
<td>145</td>
<td>224</td>
</tr>
<tr>
<td>18000</td>
<td>515</td>
<td>459</td>
</tr>
<tr>
<td>36000</td>
<td>1472</td>
<td>1333</td>
</tr>
<tr>
<td>72000</td>
<td>3432</td>
<td>3213</td>
</tr>
</tbody>
</table>

Table 1 shows the computation time and memory usage for computing the lowest 100 vibrational modes of PE particles of various sizes on a single processor NEC-SX4 (a vector processor with 2 Gflops peak performance and 8 GB shared main memory). The entire computation for extracting the lowest 100 normal modes of a 18,000-atom polyethylene particle is 20 times faster than the direct use of IRL. However, the memory usage is nearly 500 MB. For larger problems, the memory limitation on a single processor machine imposes a serious constraint on the number of atoms we can incorporate in a molecular model. The growth in memory requirements is an inevitable aspect of sparse matrix factorization (Gilbert, undated). Hence, to carry out NCA calculation for molecular systems consisting of hundreds of thousands of atoms, it is natural to consider parallelization of the overall process to utilize the large amount of memory available on distributed memory multiprocessors and NOWS.

3.3 PARALLELIZING A SHIFT-INVERT LANCZOS PROCESS

To perform a shift-invert IRL on distributed memory multiprocessor system, we must:

1. Parallelize sparse matrix factorization \( F - \sigma I = LDL^T \);
2. Parallelize matrix vector operation \( w \leftarrow (F - \sigma I)^{-1} v_j \)
   for \( j = 1, 2, \ldots, k + p \);
3. Parallelize the orthogonalization step:
   \[
   w \leftarrow w - V_j (V_j^T w), \tag{3}
   \]
   and the implicitly restarted: \( V_{k+p}^+ \leftarrow V_{k+p} Q_{k+p}, T_{k+p}^+ \leftarrow Q_{k+p}^T T_{k+p} Q_{k+p} \).

Sparse \( LDL^T \) factorization can be performed efficiently on multiprocessors using either parallel multifrontal or column-block methods. Efficient implementations depend to a large extent on using dense distributed matrix kernels and sophisticated communication schemes (Ashcraft et al. 1990, Gupta and Kumar 1994, Gupta et al. 1992, Ng and Peyton 1993, Raghavan 1997, Rothberg 1993). Such parallel implementations leave \( L \) and \( D \) distributed across processors. These factors are used to carry out the matrix vector operation \( w \leftarrow (F - \sigma I)^{-1} v_j \) by solving two sparse triangular systems:

\[
Ly = v_j \quad \text{and} \quad L^T w_j = D^{-1} y.
\]

Although the cost of a single triangular substitution is usually orders of magnitude lower than that of a sparse \( LDL^T \) factorization, this operation must be performed repeatedly in a shift-invert Lanczos iteration. Therefore, optimizing the parallel performance of a triangular substitution becomes a serious design issue in parallel NCA calculation. It is well known that the parallel performance of a conventional sparse triangular substitution is limited by the communication latency of multiprocessors. Fortunately, this performance bottleneck can be overcome by using a latency tolerant “selective inversion” technique developed earlier by Heath and Raghavan (1998), and Raghavan (1998). We postpone a brief overview of this method to Section 4 and continue with issues related to the parallelization of the orthogonalization step.

The data partitioning scheme provided by the sparse \( LDL^T \) factorization and forward (backward) substitution determines how the Lanczos basis vectors should be distributed among processors. If these vectors are permuted such that rows belonging to the same processor are grouped into a contiguous block, then the parallelization of the orthogonalization step (3) is illustrated by Figure 4. Because \( k + p \) is usually much smaller than \( n \), the tridiagonal matrix \( T_{k+p} \) and the orthogonal transformation \( Q_{k+p} \) are replicated and updated on each processor. Each processor carries out its portion of orthogonalization by performing two dense matrix vector multiplications. Communication occurs in the computation of \( V_{k+p}^+ w \). A global sum must be performed to assemble the local matrix-vector product from each processor, and the result must be broadcast to each processor for subsequent computation. This strategy is described in detail in Maschhoff and Sorensen (1996), and has been implemented in P_ARPACK using MPI (Maschhoff and Sorensen 1996).
Fig. 5 The recursive sparsity structure of \( L \) and the corresponding tree for parallel triangular solution. Shaded dense submatrices are used in distributed substitution steps and are inverted in SI (using 4 processors).

Upon the completion of the parallel Lanczos process, appropriate permutation must be applied to the local pieces of the approximate eigenvectors collected from each processor.

4 Latency tolerant sparse triangular solution

In a “shift-invert” IRL iteration, a single sparse \( \text{LDL}^T \) factorization is followed by a sequence of sparse forward and backward substitutions. Therefore, efficient parallelization of these substitution steps has a significant impact on the overall performance of NCA calculation. We now provide a brief description of the parallel sparse triangular solution and the latency-tolerant “selective inversion” method.

We will only discuss how to perform forward substitution \( Lx = b \) in parallel. A similar approach can be used to perform a backward substitution. The \( \text{LDL}^T \) factors of a sparse matrix are computed after using a nested dissection (George 1973) permutation to limit fill-in (zeros becoming nonzeros) during factorization. Such a permutation imparts a distinct sparsity structure to the factor \( L \). This structure is recursive and is a combination of zero, dense, and sparse submatrices as shown below:

\[
L = \begin{pmatrix}
L_0 & Z & L_1 \\
Z & L_0 & L_0-1 \\
M_0 & M_1 & L_0-1
\end{pmatrix}.
\] (4)

The matrices \( L_0, L_1 \) and \( L_{0-1} \) are unit lower triangular. The submatrices \( M_0 \) and \( M_1 \) are sparse whereas \( L_{0-1} \) is dense while \( Z \) is zero (and henceforth will not be shown explicitly). This sparsity structure is recursively repeated within \( L_0 \) and \( L_1 \). Such a structure allows us to divide the task of forward substitution among two processors, say \( p_0 \) and \( p_1 \). Consider partitioning the solution vector \( x \) and the right hand side \( b \) conformally with \( L \):

\[
\begin{pmatrix}
L_0 & L_1 \\
M_0 & M_1 & L_{0-1}
\end{pmatrix}
\begin{pmatrix}
x_0 \\
x_1 \end{pmatrix}
= \begin{pmatrix}
b_0 \\
b_1 
\end{pmatrix}.
\]

Now the \( x_0 \) and \( x_1 \) portion of the solution vector can be obtained independently by solving the smaller sparse triangular systems shown below on processors \( p_0 \) and \( p_1 \) respectively:

\[
L_0 x_0 = b_0 \quad \text{and} \quad L_1 x_1 = b_1 \] (5)

Finally, \( x_{0-1} \) can be computed by collecting \( x_0 \) and \( x_1 \) from \( p_0 \) and \( p_1 \), and updating \( b_{0-1} \):

\[
b_{0-1} \leftarrow b_{0-1} - M_0 x_0 - M_1 x_1 . \] (6)

The final phase of the substitution involves solving the smaller dense triangular system:

\[
L_{0-1} x_{0-1} = b_{0-1} . \] (7)

To further improve parallel efficiency, we can parallelize the update (6) and the dense triangular solve (7). This requires distributing \( L_{0-1}, x_{0-1}, \) and across \( p_0 \) and \( p_1 \).

As mentioned earlier, when \( L \) is large enough, nested dissection reproduces the above sparsity structure recur-
sively, i.e., the submatrices $L_0$ and $L_1$ can be further partitioned in the same way $L$ is partitioned. Figure 5 shows a two-level recursive structure of $L$ on a sparse matrix. The multilevel recursive scheme allows us to utilize more processors to carry out a sparse forward substitution. The hierarchical structure of the algorithm can be conveniently expressed by a binary tree as shown in Figure 5. Each subtree corresponds to a subtask of solving a smaller sparse triangular system such as $L_s x_s = b_s$. Subtrees at the same level can be processed simultaneously to produce different portions of the solution vector. The task parallelism available at lower levels of the tree is easily exploited by suitable processor to subtree assignments. At each tree node, local solutions produced by descendant subtrees are collected and used to update the right-hand side following a scheme similar to (6). The dense submatrices (shown shaded in Figure 5) are used to perform distributed triangular solution. The size of such dense matrices is typically larger at levels closer to the root.

If a large number of processors are used, the entire forward substitution process is dominated by time to compute a sequence of distributed, dense triangular solutions. Solving dense triangular systems efficiently on message-passing multiprocessors is known to be a difficult problem. Even with the use of sophisticated pipelining techniques, very large matrices are required to achieve good efficiency (Eisenstat et al. 1998, Heath and Romine 1988, Li and Coleman 1988, 1989) and the dense submatrices within a sparse $L$ are typically not large enough. The “selective inversion” (SI) technique (Raghavan 1998, Heath and Raghavan 1998) relies on inverting selected dense submatrices within the factor $L$ to replace parallel substitution by efficient parallel matrix-vector multiplication. Issues of numerical stability can be addressed by observing that numerically stable blocked and non-blocked algorithms for the inversion of dense triangular matrices can be used (Du Croz and Higham 1992). SI requires no more storage than for $L$ because selected dense triangular submatrices are inverted in place. There is, however, an added cost for performing the inversion. For sparse matrices from two and three dimensional finite-difference and finite-element meshes, the extra cost of inversion is analytically shown to be less than 5% of the cost of factorization. This extra arithmetic work does not significantly affect the total time when the factors are used to solve for a sequence of right-hand-size vectors (as in our application). In Figure 5, submatrices of $L$ that are inverted for a 4-processor implementation using SI are shown fully shaded.

Extensive empirical results in Raghavan (1998) and Heath and Raghavan (1998) demonstrate that SI is superior to schemes based on substitution when the message latency is high. In a distributed substitution scheme for a $k \times k$ dense matrix, a processor sends or receives at least one floating point value in each of the $k$ steps and therefore incurs the message start-up time at least $k$ times (it is actually much more than that, see Heath and Romine (1988) for detailed analyses). On replacing a substitution scheme by a dense matrix vector multiplication the number of messages is reduced to a single “gather” operation of a $k$ word message. This has clear benefits for performance when the message start-up time is large. Another advantage is that the arithmetic work is shared evenly by all processors whereas in a substitution scheme not all processors are involved for the first $p$ steps. As reported in the next section, we observed significant performance gains for our parallel NCA application using SI.

5 Performance

In this section, we report on the performance of our parallel NCA application on an IBM RS/6000 SP system maintained by the Computer Science and Mathematics Division at Oak Ridge National Laboratory. The system consists of 184 “Winterhawk-II” thin nodes, each equipped with four 375 MHz Power3-II processors, 2GB shared memory. Each processor contains 8 MB cache, and has the capability of performing four floating point operations simultaneously achieving a peak performance of 1.5 Gflops. Processors within each node are connected by a 128-bit, 100 MHz 6xx bus system that provides a theoretical peak communication bandwidth of 1.6 GB/sec. Communication between nodes is accommodated by high speed SP switch MX2 adapters that provide a peak bandwidth of 400 MB/second.

We performed NCA calculations for polyethylene particles and crystals of various sizes. For both problems, a threshold of $r = 14$ Å is used to exclude non-bonded interaction between atoms far apart. The difference in the geometric configuration of these molecules leads to quite different sparsity patterns of the $F$ matrix and its lower triangular factor $L$. It is not difficult to see that this structure difference is caused mainly by the non-bonded interactions between atoms.

The lattice structure of the crystal (Figure 2) suggests that each atom interacts with a fixed number of non-bonded atoms within a spherical volume with radius $r = 14$ Å. The relative positions of these non-bonded “neighbors” are the same for each atom. Consequently, the force calculation follows a natural 3-D stencil similar to the one used in a finite difference discretization. As a result, $F$ has a regular banded structure (Figure 7).

For a polyethylene particle, there appears to be no clear pattern that relates the position of each atom along a polymer chain to the existence of a non-bonded interaction between two atoms. In particular, one end of a
polymer chain may coil back to the other end (to achieve a minimum potential energy configuration), making atoms furthest apart along the chain become non-bonded “neighbors”. Thus both the number and the relative positions of the non-bonded “neighbors” can be different for different atoms. This type of variation in non-bonded interactions among different atoms yields an $F$ matrix that has an irregular nonzero pattern.

We are interested in 100 normal modes associated with the lowest vibration frequencies. We placed the target shift at $\sigma = -0.01$ in all of our calculations. This choice of shift makes $F-\sigma I$ positive definite.

Row and column permutations are applied to $F-\sigma I$ prior to the factorization phase of our NCA calculation to limit the non-zero fills in $L$. Our previous experience on the SX-4 indicated that a nested dissection ordering scheme yields the least amount of fills compared to other ordering schemes. Hence, a nested dissection ordering scheme based on domain and node-separator partition (Raghavan 1999) is used exclusively in our computa-
The reordered matrices for the 3000-atom particle and crystal are shown in Figure 6 and Figure 7 respectively.

Table 2 lists the nonzero counts of both $F$ and $L$, the number of floating point operations (flops) required to perform $LDL^T$ factorization and the (worst case) additional flops required to perform SI. As indicated by numbers in the last column, the additional operations to perform SI are no more than a small percentage of the number of operations required for an $LDL^T$ factorization. The number of nonzeros in $L$ depends on the effectiveness of fill-reducing ordering strategies applied prior to a factorization. These ordering methods can be analyzed using graph theory and it has been established that sparse matrices which are not well separable incur larger fill-in (Gilbert, undated). Fill incurred for such problems could vary (by as much as 10–20%) depending on the ordering algorithm used. However, the variation will not be significant as a function of the matrix dimension, i.e., when a sparse matrix is not well separable, all ordering methods will produce factors that are effectively dense. Based on our initial study of such graph-theoretic properties, we conjecture that the $F$ matrices associated with PE particles are not effectively separable, thus leading to significantly more fill-in than for matrices associated with PE crystals. Our experiments with different ordering methods for matrices of PE particles did not yield a fill-reducing permutation with substantially sparser Cholesky factors. From Table 2, we can see that the triangular factor associated with a PE particle contains more nonzero entries than that associated with a PE crystal, although the nonzero counts of $F$ are roughly the same for these two types of molecules. Note that the lower triangular factors of the 24,000-atom PE particle and crystal both require more than 3 GB of memory. Clearly, we cannot carry out such a large-scale NCA calculation on a single processor of the IBM SP.

We measured the elapsed time of two versions of parallel NCA calculations. The calculation that does not make use of SI is labeled as “SI-off”, and the SI-enabled calculation is labeled as “SI-on”. The measurements presented in the following sequence of tables allow us to (i) assess the parallel efficiency and the impact of SI in NCA calculation and, (ii) quantify the speedup obtained from parallelization of NCA.

The IRL parameter, $k$, which specifies the number of desired normal modes, is set to 100. We experimented with a number of choices of $p$ (number of shifts applied in each IRL restarting cycle), and found that $p = 30$ generally gives better performance in terms of the average number of sparse triangular substitutions required for each eigenvalue. The use of a larger $p$ value can lead to fewer IRL restarting cycles, but each cycle will consume more sparse triangular substitutions. We should also point out that the values of $k$ and $p$ can be adjusted dynamically during the course of IRL iterations. For example, $k$ can be set to a smaller number to allow more shifts (a larger $p$ value) to be applied at the beginning of the IRL iterations. It must ultimately be increased to the number of desired eigenvalues.

All calculations successfully returned the lowest 100 vibrational modes of the PE particles and crystals listed in Table 2 with less than 50 IRL restarting cycles. The relative residuals of all computed eigenpairs are on the order of $10^{-15}$. For 3000-atom molecules, the amount of work involved in the root-level submatrix (see Figure 5) is so small that the benefit of finer grained parallelization becomes minuscule after a few levels of dissection. Therefore, for these molecules, we only experimented with up to 16 processors. By contrast, the 24000-atom molecules requires at least 16 processors to amortize the memory requirement of the $LDL^T$ factorization, and we have experimented with as many as 64 processors.

Table 2

| Problem | $N$ | $|F|$ (10^3) | $|L|$ (10^3) | Factor Ops (10^6) | SI Ops (10^5) |
|---------|-----|-------------|-------------|------------------|---------------|
| pp3k    | 9,000 | 1,664       | 15,515      | 32,740           | 3,895         | 11.89%       |
| pp6k    | 18,000 | 3,458       | 51,312      | 185,519          | 26,179        | 14.11%       |
| pp12k   | 36,000 | 7,128       | 148,108     | 799,285          | 85,214        | 10.66%       |
| pp24k   | 72,000 | 14,394      | 476,573     | 4,434,105        | 481,728       | 10.86%       |
| pc3k    | 9,000  | 1,450       | 9,982       | 13,174           | 880           | 6.41%        |
| pc24k   | 72,000 | 14,667      | 393,360     | 3,037,576        | 218,172       | 7.18%        |
Before presenting the observed execution times, we first consider the relative cost of each major step. The first step concerns the factorization \( F = \text{of} = \text{LDL}^T \), and it is performed once. When SI is activated, there is an additional one-time overhead of computing inverses of selected submatrices in \( L \); the flop counts shown in Table 2 indicate that this overhead is not significant. The factors are used to compute solution vectors for a sequence of right hand side vectors by substitution. The number of flops consumed in each substitution step (for both the “SI-on” and “SI-off” versions of the calculation) is equal to approximately four times the number of nonzeros in the factor \( L \). From Table 2 we can see that the cost of a single substitution is several orders of magnitude smaller than that of factorization and selective inversion combined. However, when multiplied by the number of triangular substitutions required in IRL, the cost becomes comparable, and in some cases even exceeds that associated with the factorization (as shown in Table 3).

Because the dimension of the Krylov subspace maintained by IRL is considerably smaller than the size of \( F \), and because no data redistribution is required in the orthogonalization process, the relative cost of orthogonalization is practically negligible and limited to under 2–3% of the overall cost. In presenting our timing results, we therefore focus more on the differences between the “SI-on” and “SI-off” versions of the factorization and substitution steps.

In Table 4, we compare the elapsed time associated with a standard multifrontal factorization without using SI with the total elapsed time associated with a SI-enabled factorization. We observe that the standard multifrontal factorization scales well from 4 to 64 processors for most of PE particles and crystals. The speedup is nearly perfect as the number of processors is increased. In some cases (e.g. \( pp24k \) and \( pc24k \)), superlinear speedup is observed when the number of processors is increased from 32 to 64. We conjecture this is due to reduction in cache misses as the required memory per processor decreases upon increasing the number of processors. Because of the relatively small number of nonzeros in the \( F \) matrix corresponding to the 3000-atom particle and crystal, only a modest speedup of 1.3–1.4 is achieved when the number of processors is increased from 8 to 16. The timing results in Table 4 also indicate that it takes longer to perform SI-enabled factorization because of the SI overhead. However, scalability remains satisfactory.

In contrast to the \( \text{LDL}^T \) factorization, obtaining an ideal parallel speedup for sparse forward and backward substitution is more difficult. When the traditional sparse substitution scheme is used, the reduction in elapsed time is far below 50% as the number of processors is doubled. This is shown in Table 5. In some cases (\( pp3k \), \( pc3k \)), elapsed time even increases as we increase the number of processors. Clearly, the communication overhead incurred in solving smaller but dense intermediate triangular systems prevented us from achieving satisfactory parallel performance. This is not surprising, as distributed substitution schemes are known to be non-scalable.

From Table 5, it is clear that the with the help of SI, the solution time is improved dramatically. Moreover, as the solution process becomes more efficient, the overall scalability is also improved. For the 24,000-atom particle and crystal system, the speedup is nearly ideal.

### Table 3
The flops consumed by the “SI-on” and “SI-off” versions of the \( \text{LDL}^T \) factorization are shown along with the number of substitution steps and the total substitution flop counts. The latter is shown in the rightmost column as a percentage of the flop count associated with the “SI-off” version of the factorization.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Factorization</th>
<th>Factorization and SI</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SI-off ( 10^6 )</td>
<td>SI-on ( 10^6 )</td>
<td>Count</td>
</tr>
<tr>
<td>( pp3k )</td>
<td>32,740</td>
<td>36,635</td>
<td>605</td>
</tr>
<tr>
<td>( pp6k )</td>
<td>185,519</td>
<td>212,238</td>
<td>888</td>
</tr>
<tr>
<td>( pp12k )</td>
<td>799,285</td>
<td>884,499</td>
<td>1,185</td>
</tr>
<tr>
<td>( pp24k )</td>
<td>4,434,105</td>
<td>4,915,833</td>
<td>450</td>
</tr>
<tr>
<td>( pc3k )</td>
<td>13,174</td>
<td>14,054</td>
<td>240</td>
</tr>
<tr>
<td>( pc24k )</td>
<td>3,037,576</td>
<td>3,255,748</td>
<td>300</td>
</tr>
</tbody>
</table>
### Table 4
A comparison of the elapsed time (in seconds) associated with the “SI-on” and “SI-off” versions of the sparse $LDLT$ factorization

<table>
<thead>
<tr>
<th>Problem</th>
<th>SI</th>
<th>Number of processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>$pp3k$</td>
<td>off</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>32</td>
</tr>
<tr>
<td>$pp6k$</td>
<td>off</td>
<td>196</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>238</td>
</tr>
<tr>
<td>$pp12k$</td>
<td>off</td>
<td>1318</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>1613</td>
</tr>
<tr>
<td>$pp24k$</td>
<td>off</td>
<td>2723</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>3054</td>
</tr>
<tr>
<td>$pc3k$</td>
<td>off</td>
<td>8.4</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>8.5</td>
</tr>
<tr>
<td>$pc24k$</td>
<td>off</td>
<td>2111</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>2254</td>
</tr>
</tbody>
</table>

### Table 5
The elapsed time (in seconds) associated with the “SI-on” and “SI-off” versions of the sparse forward and backward substitution phase of the NCA calculation

<table>
<thead>
<tr>
<th>Problem</th>
<th>SI</th>
<th>Number of processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>$pp3k$</td>
<td>off</td>
<td>410</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>86</td>
</tr>
<tr>
<td>$pp6k$</td>
<td>off</td>
<td>1738</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>374</td>
</tr>
<tr>
<td>$pp12k$</td>
<td>off</td>
<td>5517</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>1884</td>
</tr>
<tr>
<td>$pp24k$</td>
<td>off</td>
<td>3875</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>864</td>
</tr>
<tr>
<td>$pc3k$</td>
<td>off</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>23</td>
</tr>
<tr>
<td>$pc24k$</td>
<td>off</td>
<td>2139</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>484</td>
</tr>
</tbody>
</table>
To put a more precise quantification on the performance trade-offs between using and not using SI, we consider in greater detail the timing results for the the 24,000-atom particle and crystal models. We define $\delta$ to be the relative overhead introduced by SI. It is computed as:

$$
\delta = \frac{T_2}{T_1} - 1,
$$

where $T_1$ and $T_2$ are the elapsed time for the standard multifrontal $LDL^T$ and the SI-enabled $LDL^T$ factorizations respectively. The observed overhead values are shown in Table 6. They range from 5% to 12% depending on the number of processors used and the amount of fill-in in $L$. This observation is consistent with the maximum overhead indicated by raw operation counts listed in Tables 2 and 3. The overhead operation count increases

---

**Table 6**

Overhead and improvement ratios $\delta, \rho$ for the two largest matrices for Scheme 2 of NCA.

<table>
<thead>
<tr>
<th>Processors</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SI overhead ($\delta$)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pp24k</td>
<td>12%</td>
<td>5%</td>
<td>4%</td>
</tr>
<tr>
<td>pc24k</td>
<td>6%</td>
<td>5%</td>
<td>5%</td>
</tr>
<tr>
<td><strong>Triangular solution improvement ($\rho$)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pp24k</td>
<td>449%</td>
<td>623%</td>
<td>813%</td>
</tr>
<tr>
<td>pc24k</td>
<td>442%</td>
<td>679%</td>
<td>824%</td>
</tr>
</tbody>
</table>

---

**Fig. 8** The elapsed time associated with each phase of both the “SI-off” (1) and “SI-on” (2) versions of the parallel NCA calculation performed to obtain the lowest 100 vibrational modes of the $pp12k$; the figure to the right shows the time for each phase as a percentage of the total time.
with the number of processors but not linearly, as indicated by the analysis in Raghavan (1998). Typically, the relative increase in overhead operations in scaling from 16 to 32 processors is smaller than the relative increase in scaling from say 4 to 8 processors. Consequently, as shown in Table 6 we observe a reduction in SI overhead time when the number of processors is increased. This one-time overhead directly improves the performance of triangular solution required in IRL. We compute an improvement ratio:

\[ \rho = \frac{T_{s1}}{T_{s2}} - 1, \]

where \( T_{s1} \) and \( T_{s2} \) are elapsed time measurements for all triangular solution steps in the “SI-off” and “SI-on” versions of the NCA calculation respectively. The values of \( \rho \) in Table 6 indicate that the improvement is indeed quite significant and well worth the relatively small overhead.

Figure 8 shows the percentage of elapsed time consumed by each phase of the NCA calculation. The “SI-off” version is marked by 1, and the “SI-on” version is marked by 2. We use observed time measurements for pp12k, the 12,000-atom particle, because it covers the range of processor sizes. As mentioned earlier, the cost of orthogonalization is relatively low (2–3% of the entire computation). Due to the large number of triangular substitutions performed in IRL (see Table 3), the elapsed time spent in the triangular substitution phase is almost as much as that spent in the \( LDL^T \) factorization phase when SI is used. Without SI, the triangular substitution phase dominates the entire calculation.

Because triangular substitutions constitute a large portion of the NCA calculation, the performance improvement contributed by SI leads to a significant improvement in the overall performance of the normal mode calculation. In Figure 9, we plot the elapsed time for computing the lowest 100 vibrational modes of the pp12k molecule against the number of processors used. Clearly, the SI-enabled version of the calculation performs much better than the “SI-off” version. Reasonable parallel speedup is achieved with up to 64 processors.

6 Concluding Remarks

We have developed an efficient parallel computational scheme to carry out large-scale normal coordinate anal-
ysis on distributed memory multiprocessors. A shifted and inverted version of the implicitly restarted Lanczos (IRL) algorithm (Sorensen 1992) is employed to seek a small number of eigenvalues and corresponding eigenvectors of a large sparse force constant matrix. The “shift-invert” operation is carried out by using a parallel multifrontal sparse linear equation solver that makes use of the “selective inversion” technique (Raghavan 1998) to reduce communication latency. Although our original motivation for parallelization was to overcome memory limitations on traditional sequential or shared memory systems, our computational experiments indicated that scalable performance can be achieved on systems with a relatively large number of processors.

When a large number of normal modes are needed, it may be advantageous to combine several “shift-invert” IRL runs, each responsible for computing a subset of the desired modes, to maintain a rapid convergence rate of IRL. Because a different shift is used in each “shift-invert” IRL run, additional sparse \( \text{LDL}^T \) factorizations must be performed. To minimize the overall computational cost, heuristics must be developed to determine when to terminate a particular IRL run and how to select a new target shift to begin the next IRL run. Furthermore, we must ensure that converged eigenvalues are not recomputed or recounted, and no eigenvalue is missed in successive IRL runs. Some of these issues were discussed in Grimes et al. (1994) and Yang (1998). Further research is needed in the context of distributed parallel environment.

We should point out that it is possible to replace a Lanczos type of algorithm with the Jacobi-Davidson (JD) ( Sleijpen and van der Vorst 1996) algorithm in NCA calculation to eliminate the need to perform sparse \( \text{LDL}^T \) factorizations and triangular substitutions. The JD algorithm can be viewed as an inner-outer iteration scheme. In a single step of the JD algorithm, approximations to the desired eigenpairs are extracted from a projection subspace by applying the Rayleigh-Ritz procedure in the outer iteration. A preconditioned inner iteration is used to solve an eigensolver correction equation. The approximate correction vector can be added into the existing orthonormal basis of the projection subspace required in the outer iteration. The JD approach was pursued in Yang et al. (2001) for NCA calculation with some success. However, no parallel implementation is available yet because the incomplete Cholesky factorization preconditioner used in Yang et al. (2001) has not been effectively parallelized. A number of practical issues such as the convergence criterion for the inner iteration remains to be investigated further. Unlike the “shift-invert” Lanczos algorithm, there is no simple mechanism to verify whether one has captured all the desired normal modes in JD. This difficulty makes JD less attractive when the “shift-invert” operation can be carried out efficiently.

**ACKNOWLEDGEMENT**

We would like to thank the anonymous referees for carefully reading the manuscript and providing us with helpful comments.

**REFERENCES**


Gilbert, J.R. *Graph Separator Theorems and Sparse Gaussian Elimination*. PhD thesis, Department of Computer Science, Stanford University.


Raghavan, P. 1999. DSCPACK: A Domain-Separator Cholesky PACKage. Solver for symmetric positive definite systems on multiprocessors and NOWs using C and MPI. Package has a two stage parallel nested dissection, higher-level BLAS, fast repeated solves and an easy to use parallel interface. Available upon request.


