*Computers & Chemistry* Vol. 10, No. 3, pp. 165–173, 1986
Printed in Great Britain

0097-8485/86 $3.00 + 0.00
Pergamon Journals Ltd

# IMPLEMENTATION OF A MACROMOLECULAR MECHANICS PROGRAM ON A CYBER 205 SUPERCOMPUTER†

STEVEN L. GALLION, RONALD M. LEVY‡, PAUL K. WEINER and FUMIO HIRATA
Department of Chemistry, Rutgers University, New Brunswick, NJ 08903, U.S.A.

(*Received* 8 *August* 1985)

**Abstract**—We describe the steps involved in the conversion and vectorization of a macromolecular mechanics program on a CYBER 205 supercomputer. Several programming problems that we have encountered in optimizing a molecular mechanics program for a CYBER 205 are discussed including: the effect of vector length on speed, design of alternative algorithms, the migration of bottlenecks and rate differences obtained for logical vs arithmetic operations. The vectorized code for particular subroutines runs more than 200 times faster on the CYBER 205 as compared with a VAX 11/780. Using an enzyme-inhibitor complex, *Rhizopus Chinensis* carboxyl protease-pepstatin, containing 2,844 atoms as a model for this type of calculation, we have obtained an overall increase in speed of between 80–120 over a VAX 11/780 equipped with floating point accelerator for the evaluation of the conformational energy and forces on the atoms.

## INTRODUCTION

The use of Class VI supercomputers is increasing very rapidly. Advances in networking hardware and software have greatly facilitated program development on these machines which can now be accessed directly from departmental minicomputers. The machine used for computational chemistry at Rutgers University for the past several years has been a departmental VAX 11/780. Our research group has carried out large scale calculations concerned with the structural and dynamical features of biological macromolecules (Levy & Keepers, 1985). The methods used include: molecular mechanics, molecular dynamics and classical and quantum Monte Carlo simulations. As an example of the size of these calculations, we note that a 300 ps molecular dynamics simulation of the protein myoglobin in vacuo required 1200 CPU hours on our VAX 11/780 equipped with floating point accelerator (Levy *et al.*, 1985). The results of a detailed analysis of this protein simulation demonstrated the need for longer trajectories and a more realistic treatment of the protein environment. The size of such calculations are clearly beyond the capabilities of the VAX 11/780. We have therefore turned to the use of supercomputers. Although general descriptions of the potential speed of supercomputers for scientific applications have appeared (Hockney *et al.*, 1981; Levine, 1982), it is not yet clear how well complex programs of the kind used by computational chemists will perform on these machines. Reports concerning the peformance of electronic structure programs on supercomputers have recently appeared (Ahlrichs *et al.*, 1985; Rappe *et al.*, 1985). In this article we describe the implementation and vectorization of a macromolecular mechanics program on a CYBER 205 supercomputer. We review the steps involved in conversion and optimization of the code, and present the methods used to vectorize the most time consuming portions of the calculations. Several programming problems encountered in optimizing a molecular mechanics program for a CYBER 205 are discussed including the migration of bottlenecks, alternative vectorization schemes and the effect of logical versus arithmetic operations on speed.

The program modifications and vectorization described here were performed on the CYBER 205 at Purdue University. This machine and its front end VAX 11/780 at Purdue are accessed directly from the chemistry VAX 11/780 at Rutgers via ARPANET using TCP/IP network protocol. This means that we are able to log on to the Purdue site as a remote terminal, use the same editor at Purdue and on our chemistry VAX, and accomplish file transfer of programs and data, rapidly and efficiently. The network access has greatly aided the task of program conversion and vectorization of code using CYBER 200 Fortran.

## THE PROBLEM

The ability to evaluate the energy of a molecular system as a function of the atomic coordinates is fundamental to all theoretical efforts to understand chemical stability, kinetics, and dynamics at a molecular level. For biological macromolecules including proteins and nucleic acids, the system size necessitates the use of empirical energy functions of the molecular mechanics type. These functions contain terms describing covalent bond interactions, bond angle interactions, torsional angle interactions and nonbonded (van der Waals, hydrogen bond, and Coulomb) interactions. The macromolecular mechanics program we

†This work has been supported in part by grants from the National Institutes of Health (GM-30580) and the National Science Foundation Office of Advanced Scientific Computing (DMB 8541242/8306023). R.M.L. is an Alfred P. Sloan Foundation Fellow and the recipient of an NIH Research Career Development Award.

‡Author to whom correspondence should be addressed.

have implemented on the CYBER 205 is a version of the program package AMBER written by Paul Weiner and Peter Kollman (Weiner & Kollman, 1981).

We use as an example throughout this paper, molecular mechanics calculations we are carrying out on the protein *Rhizopus Chinensis* carboxyl protease. The x-ray structure of this enzyme has been solved with an inhibitor, pepstatin, in the active site (Bott *et al.*, 1982). The system provides a model of a key enzyme (Renin) involved in the regulation of blood pressure and there is a large amount of thermodynamic data available concerning substrate binding. This macromolecular complex contains 2488 atoms, 2893 bonds, 4161 bond angles, and 6447 torsional angles. The bulk of the computer time on the VAX required to evaluate the energy of this protein for a particular conformation is spent in the evaluation of the nonbonded energy. The nonbonded terms must be evaluated between all pairs of "nonbonded" atoms, that is atoms which do not share a common bond, bond angle or torsion angle. For a molecule with $N$ atoms this requires the evaluation of approximately $N(N\text{-}1)/2$ terms. To cut down on the computation time, a cutoff radius $R_c$ is conventionally employed, beyond which the pairwise atomic interactions are neglected. One method for evaluating the nonbonded energy involves the generation of a list of nonbonded atom partners which is updated periodically (typically, every ten energy evaluations) as the conformation of the protein changes. For atomic systems which contain less than ~6000 atoms the nonbonded pair list technique for evaluating the energy appears to be faster than alternative grid search methods (van Gunsteren *et al.*, 1984). Our effort has been directed towards vectorizing the most time consuming subroutines including those dealing with the generation of the nonbonded pair list, the evaluation of the nonbonded energy and the evaluation of the torsional energy.

## PROGRAM STRUCTURE FOR THE CYBER 205

The CYBER 205 contains both a high speed scalar processing unit and vector processors, called vector pipelines. A vector machine instruction operates on an array of values much more efficiently than repeated scalar instructions for operations on individual values. Most of the effort in optimizing code for the CYBER 205 involves recoding DO-loops to take advantage of the vector instruction set. The CYBER 205 can only process contiguous arrays so that special CYBER 205 hardware routines must be used to "gather" elements into and "scatter" elements from contiguous regions of memory, the Q8 routines. Bit vectors are used to control the flow of elements to and from contiguous regions of memory. Furthermore, the CYBER 205 vector processor operates most efficiently on long vector lengths. Since the "start-up" time of the vector pipeline on the CYBER 205 is relatively long, vectors must be large before the vector processor works optimally. In contrast to the CRAY-1 which processes vectors with a maximum size of 65, the CYBER 205 can process arrays with a maximum length of 65,535 elements. The CYBER 205 at Purdue is a two pipe machine which has a peak efficiency of 100 MFLOPS when operating on 64 bit operands. The CYBER 205 architecture is memory to memory, that is vectors are stored in memory, moved through pipes and returned to memory without encountering a register. The process of paging in new data from virtual address space interrupts the pipeline. To obtain peak efficiency it is important to prevent paging. The MFLOP rate as a function of vector length for a simple arithmetic addition is shown in Table 1. For vector lengths of 100 the machine performs at 35% peak speed whereas for a vector length of 3,500 the machine performs at 95% peak speed. The effect of paging is evident as the MFLOP rate decreases when vector lengths greater than 10,000 are used with small pages (2048 words). When using large pages (65,536 words) the machine asymptotically approaches its maximum speed as the vector length is increased to the largest value (see Table 1). On the Purdue CYBER, the main memory capacity is 28 large pages; as the memory resources requested for a job approach this limit, the job priority decreases and turnaround time increases greatly. Thus, when recoding a subroutine to take advantage of the vector processor the choice of vector length is constrained by paging and scheduling considerations.

In the scalar macromolecular mechanics code, the generation of the neighbor list and the evaluation of the nonbonded energy require a double DO-loop. The most straightforward approach to recoding these subroutines is to vectorize the inner DO-loop. This was tried first and proved to be too slow. The inner loop of the nonbonded energy calculation is too small to allow for optimal vector performance on the CYBER 205 so that the algorithm had to be redesigned to lengthen the vector size. On each pass through the DO-loop for the optimized vector code, all nonbonded interactions are calculated for blocks of atoms at one time. The number of nonbonded pairs in each block is adjusted to achieve optimum performance. Paging and scheduling considerations discussed below are important in determining the block size. The vectorized code for the neighbor list and nonbonded energy subroutines are given in Appendixes A and B.

The generation of the neighbor list requires the formation of two arrays IAR1 and IAR2. IAR1 serves as a pointer into IAR2. IAR1 (I) is the total number of nonbonded partners of atom I, and IAR2 contains the atom numbers of the nonbonded partners of atom I. These arrays are formed by searching from atom I+1 to $N$ for all atoms which lie within a distance $R_c$ from atom I and are also not excluded atoms. This procedure is repeated N-1 times.

The simple vectorization of the inner DO-loop of the nonbonded energy subroutine led to poor performance of the vector code when compared with the scalar version of this subroutine. We found that for a cutoff radius of 8Å the average number of nonbonded partners is ~50; this is then the vector length for the algorithm. There are 122,379 nonbonded pairs for

Table 1. Effect of vector length on the number of floating point operations per second

| Vector Length | Floating Point Operations Per Second × $10^{-6}$(MFLOPS) | |
|---|---|---|
| | Small Page | Large Page |
| 65,536 | 74.5 | 99.7 |
| 35,000 | 77.6 | 99.4 |
| 20,000 | 79.0 | |
| 10,000 | 97.9 | |
| 3,500 | 94.5 | 94.8 |
| 1,000 | 83.2 | 83.6 |
| 500 | 72.6 | |
| 250 | 55.7 | |
| 100 | 34.6 | |
| 50 | 20.1 | |
| 25 | 11.1 | |
| 10 | 4.8 | |

which the energy must be evaluated with an 8Å cutoff for the system studied. Since the maximum vector length on the CYBER 205 is 65,535 the optimized algorithm requires at least two iterations of the DO-loop. However, when vector lengths approaching the maximum value are used, paging and scheduling problems decrease the efficiency of the program. After preliminary studies we chose to introduce a block structure which created vector lengths close to 3,500. This length was chosen because the arrays used in all vectorized subroutines could then be mapped onto ten large pages. Furthermore, for simple arithmetic operations, 95% of the maximum MFLOP rate is achieved with a vector length of 3,500 (Table 1).

## RESULTS AND DISCUSSION

The results of timing studies for the evaluation of the conformational energy of *Rhizopus Chinensis* carboxyl protease-pepstatin complex using an 8Å cutoff for the generation of the nonbonded (NB) list are given in Table 2 along with reference values for the VAX 11/780. We first compare timing on the VAX with the results obtained for the same scalar code on the CYBER 205. The time required to generate the NB list on the VAX is 204.4 seconds, whereas the scalar code on the CYBER 205 takes 12.3 seconds. Thus, simply using the fast scalar processor on the CYBER provides a factor of 16 increase in speed for the NB list generation. For the NB energy evaluation the increased speed using the CYBER scalar processor over that of the VAX is even greater; the time required for one energy evaluation is 37.6 seconds on the VAX and 1.04 seconds on the CYBER. It should be noted that the rate enhancement varies between 16 to 35 for the different subroutines when comparing the scalar performance of the CYBER 205 with the VAX (Table 2, columns 1 and 2). In general, those subroutines which contain more floating point operations as compared with logical operations exhibit greater rate enhancements. One complete energy evaluation (excluding the NB list generation) for this 2,488 atom system takes 53.74 seconds on the VAX and 1.88 seconds on the CYBER 205 resulting in an overall speed increase of 29 when using the CYBER 205 as a scalar processor for this calculation.

Table 2. Results of CYBER 205 timing studies[a]

| Routine | VAX | Scalar 205 | Vector[b] | Vector[c] | Final[d] |
|---|---|---|---|---|---|
| NB List | 204.4 | 12.3 | 1.750 | 1.750 | 1.750 |
| NB Energy | 37.6 | 1.04 | 0.851 | 0.358 | 0.282 |
| Dihedral | 10.78 | 0.542 | 0.541 | 0.540 | 0.051 |
| 1-4 NB | 2.10 | 0.117 | 0.117 | 0.117 | 0.072 |
| Hbond | 1.84 | 0.056 | | | |
| Bond | 0.56 | 0.022 | | | |
| Angle | 2.07 | 0.101 | | | |
| Total[e] | 53.74 | 1.88 | 1.66 | 1.35 | 0.68 |

[a]The times listed are in seconds for molecular mechanics calculations on the 2,844 atom system *Rhizopus Chinensis* carboxyl protease plus inhibitor pepstatin. The Hbond, Bond and Angle subroutines have not been vectorized.

[b]Timing corresponding to explicit vectorization of the inner DO loop of the NB list and NB Energy subroutines.

[c]Timing after recoding the NB Energy subroutine to form long vectors (3,500) by introducing a block structure.

[d]The final timing corresponds to explicit vectorization of the inner DO loop of the NB list subroutine, the introduction of a block structure to form long vectors in the NB Energy and Dihedral subroutines and the mapping of large arrays to large pages.

[e]Total corresponds to the time for one complete energy evaluation including all terms in the molecular mechanics potential, and the evaluation of the forces on the atoms but excluding the generation of the NB pair list.

The timing results obtained for the initial inner DO-loop vectorization of the NB list and NB energy are presented in Table 2, column 3. The vectorized version of the NB list generation requires 1.75 s as compared with 12.3 s for the scalar CYBER 205 version; for a rate enhancement of 7 through vectorization of the code. In contrast, the increase in speed obtained by vectorizing the NB energy evaluation is only 20% (1.04 s CYBER 205 scalar code; 0.85 s CYBER 205 vector code). The poor performance of the code obtained by vectorizing the inner DO-loop of the NB energy subroutine led to the development of the block structure required to lengthen the vectors used in this subroutine. Upon introduction of this block structure, the time for the NB energy evaluation decreased to 0.358 s (Table 2, column 4). This is the time for one pass through the entire NB subroutine. It includes the time required to evaluate the Van der Waals and electrostatic energies and forces, to gather all indices for the nonbonded arrays and to perform the logical operations associated with the cutoff criteria. The time required to evaluate the conformational energy is 1.35 s and the most time consuming part of the molecular mechanics energy calculation now involved the evaluation of the torsional energy. Thus the bottleneck migrated from the NB energy to the torsional subroutine.

In the last column of Table 2 we present the final timing results. The Dihedral subroutine has been vectorized using long (3,500) vectors by introducing a block structure similar to the one developed for the NB energy subroutine and mapping all arrays to large pages. The final vectorized version of the Dihedral subroutine runs more than 200 times faster on the CYBER 205 than the VAX 11/780. This is the greatest increase in speed we have obtained upon vectorization of a subroutine. The time required to evaluate the total energy is 0.68 s. With an 8Å cutoff, the evaluation of the conformational energy of the protein is ~80 times faster on the CYBER 205 compared with the VAX 11/780. The vectorized version of AMBER on the CYBER 205 appears to be about twice as fast as a similar macromolecular mechanics program running on a CRAY-1 (van Gunsteren *et al.*, 1984). This may be related to the ability to create long vectors for these calculations which makes it possible to take advantage of the CYBER pipeline architecture (Vogelsang *et al.*, 1983). Concerning the final timing results in Table 2, it should be noted that approximately 25% of the time required to evaluate the conformational energy is spent in subroutines which have not yet been vectorized. While these subroutines (Bond, Angle, and Hydrogen bond) did not take up a significant amount of time in the scalar version of the program, they must be vectorized to achieve an additional rate enhancement. It is clear that after the most time consuming subroutines have been vectorized, a proportionally greater increase in effort is required to obtain the next increment in speed.

We have studied the effect of varying the cutoff radius used to generate the nonbonded neighbor list on the speed of the conformational energy calculations. Timing runs have been performed for values of $R_c$ between 8Å and 14Å. Increasing $R_c$ provides a measure of the efficiency of the vector code for larger systems, e.g. a solvated protein. 122,379 nonbonded atom pairs are generated for our model system with an 8Å cutoff; the neighbor list increases to 557,414 nonbonded atom pairs when $R_c$ is increased to 14Å. For comparison, the simulation of the crystal dynamics of a unit cell for the protein required the evaluation of 226,000 nonbonded pairs at each step (van Gunsteren *et al.*, 1983). Figure 1 shows the time required to evaluate the total energy on the VAX 11/780 and on the CYBER 205 as a function of cutoff radius. With an 8Å cutoff the vectorized code on the CYBER is 80 times faster than the scalar code on the VAX 11/780; with a 14Å cutoff the calculation on the CYBER is ~120 times faster than the VAX. The increase in the relative speed of the CYBER when compared with the VAX for the larger calculation ($R_c$ = 14Å) is due to an increase in the time spent in the vector processor relative to scalar processor on the CYBER which makes the CYBER more efficient. Paging on the VAX also contributes to the effect.

The CYBER 205 has gained a reputation as a difficult machine to program, particularly in comparison to the CRAY-1. It is of interest therefore to provide a summary of the time and programming effort required to achieve the results reported in this paper. The programming effort began in January, 1985 when one of us (SLG) attended a CYBER 205 programmer training course at Purdue University. For the calculations reported in this paper about 5,000 lines of code are resident on the CYBER 205; several preprocessors required for the calculation are run on a VAX. It took about one week to successfully compile and run the

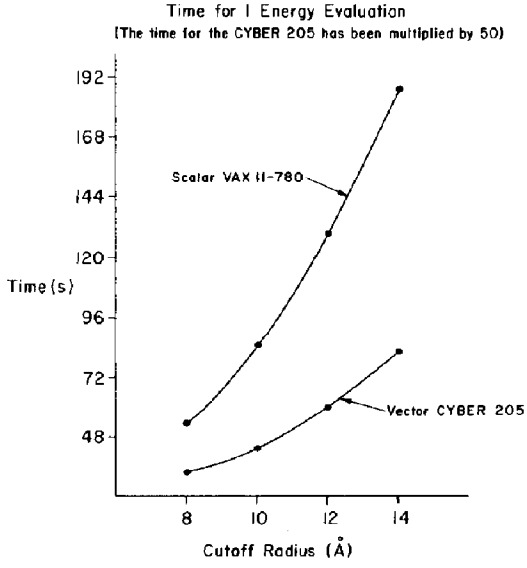


Fig. 1. Total time for one energy evaluation as a function of the cutoff radius used to generate the nonbonded pair list. The number of nonbonded pairs included in the list and cutoff radii used for these calculations are: (122,379 pairs, $R_c$ = 8Å), (229, 127 pairs, $R_c$ = 10Å), (375,742 pairs, $R_c$ = 12Å), (557,414 pairs, $R_c$ = 14Å).

code in scalar mode on the CYBER 205. The initial vectorization of the inner DO-loops of the NB list and NB Energy subroutines took one month to complete. The recoding of the vectorized NB Energy subroutine involving the introduction of a block structure required an additional month. Finally, the vectorization of the Dihedral subroutine which used the block structure already worked out for the NB subroutine took less than one week of program development. Not surprisingly, the initial vectorization proceeded slowly because parallel programming concepts and specialized CYBER hardware subroutines had to be mastered. However, there are only a relatively small number (~8) of specialized CYBER calls that are used repeatedly. In our experience, after an initial training period, programming on the CYBER 205 is not very much more difficult than scalar programming. The use of a friendly front end (VAX) and efficient networking to the CYBER greatly eased the programming task.

With the current vectorization of the most time consuming subroutines in the molecular mechanics program AMBER, we have increased the speed of the program between 80–120 times over that of a VAX 11/780. The actual speed increase depends on the choice of the cutoff radius used to create the nonbonded atom pair list. This increased speed will make it possible to model macromolecular systems that were not previously practical to study; for example, molecular dynamics simulations of proteins in a crystal environment and in solution, simulations of substrate–enzyme docking and energetics, and other numerically intensive calculations on large molecular systems.

*Acknowledgements*—The version of AMBER vectorized on the CYBER 205 is based upon source code supplied by P.K.W. R.M.L. thanks Peter Kollman for also providing a version of AMBER. S.L.G. thanks the staff of the Purdue University Computer Center for the courtesy extended during his visit and for continued assistance. We thank David Davies and Joel Sussman for providing the coordinates of the *Rhizopus Chinensis* carboxyl protease-pepstatin complex.

## REFERENCES

Ahlrichs, R., Bohm, H. J., Ehrhardt, C., Scharf, P., Schiffer, H., Lischka, H. & Schindler, M. (1985), *J. Comput. Chem.* **6**, 200.

Bott, R., Subramanian E. & Davies, R. E. (1982), *Biochemistry* **21**, 6956.

van Gunsteren, W. F., Berendsen, H. J. C., Colonna, F., Perahia, D., Hollenberg, J. P. & Lellouch, D. (1984), *J. Comput. Chem.* **5**, 272.

van Gunsteren, W. F., Berendsen, H. J. C., Hermans, J., Hol, W. G. J., & Postma, J. P. M. (1983), *Proc. Natl. Acad. Sci.*, **80**, 4315.

Hockney, R. W. & Jesshope, C. R. (1981), *Parallel Computers*, Hilger, Bristol, England.

Levine, R. (1982), *Scientific American* **246**, 118.

Levy, R. M. & Keepers, J. "Computer Simulations of Protein Dymanics: Theory and Experiment," Comments on Cellular and Molecular Biophysics, in press, 1985.

Levy, R. M., Sheridan, R. P., Keepers, J. W., Dubey, G. S., Karplus, M., & Swaminathan, S. (1985), *Biophys. J.* **48**, 509.

Rappe, A. K. (1985), *J. Comput. Chem.* **5**, 471.

Vogelsang, R., Schoen, M., & Hoheisel, C. (1983), *Computer Phys. Comm.* **30**, 235.

Weiner, P. K. & Kollman, P. (1981), *J. Comput. Chem.* **2**, 287.

## APPENDIX A: VECTORIZED LIST FORMATION PROCEDURE

The selection of nonbonded partners J for a particular atom I consists of:

(1) Determining which of the distances from atom I to all atoms J forward in the tree structure (i.e.; from I+1 to N) are less than the cutoff distance CUT. This information is stored in the bit vector BITDIST.

(2) Determining those atoms J which are not excluded from either nonbonded or hydrogen bonded interactions with atom I. The data structure of AMBER greatly simplifies this task. The array element IBLO(I) contains the number of exclusions for atom I and the array INB contains the actual excluded atom list. To create the bit vector BITNOTEX containing a binary '1' in the position corresponding to an atom which is not in the exclusion list for atom I, the numbers of the excluded atoms are scattered into a previously zeroed array, INDEXDYN, at positions corresponding to their respective number. The positions which are still zero after this operation then correspond to the atom numbers which are not excluded on the basis of connectivity from either the nonbonded or hydrogen bonded lists.

(3) Locating the atoms forward in the tree structure that are potential hydrogen bonding partners and subsequently excluding these from consideration in the formation of IAR2.

(4) The final list of nonbonded partners of atom I is then composed of those atoms J which are represented by a '1' in the Jth position of all three bit vectors formed in operations (1) through (3) above.

```
      subroutine list
      dimension indexdyn(3500)
      dimension ihbarray(3500),ihbindex(3500)
      dimension iaddvec(3500),distance(3500)
      dimension itemp(3500),itemp2(3500)
      bit bitnot0(3500),bitdist(3500)
      bit bitnotex(3500),bitfinal(3500),bitnothb(3500)
c
      common /carts/ x(10500),xo(10500)
     *       /cons1/ dielc,pnbfac,pelfac,ntypes,idist
     *       /hbond/ nhb,ihb(40000),jhb(40000),ich(40000)
     *       /info/ natom,ico(900),iac(3500)
     *       /nbterm/ cut,nbuck,iblo(3500),inb(15000)
     *       /select/ iar1(3500),iar2(208000)
c
        n=natom
        enb = 0.
        eel = 0.
        ehb = 0.
        one(1;3500)=1.0
        iione(1;3500)=1
        iacivec(1;n)=(iac(1;n)-iione(1;n))*ntypes
        iindex(1;10500)=q8vintl(1,1;iindex(1;10500))
        xcoor(1;n)=q8vgathp(x(1;0),3,n;xcoor(1;n))
        ycoor(1;n)=q8vgathp(x(2;0),3,n;ycoor(1;n))
        zcoor(1;n)=q8vgathp(x(3;0),3,n;zcoor(1;n))
c
        istart=1
        istart2=1
        nhbiter=1
        iii=0
        nhb=0
        do 10 i=1,n-1
        indexdyn(i+1;n-i)=0
        nx=iblo(i)
        next=nx+1
        left=n-i
        xi=xcoor(i)
        yi=ycoor(i)
        zi=zcoor(i)
        distance(i+1;n-i)=
     .       ((xcoor(i+1;n-i)-xi)*(xcoor(i+1;n-i)-xi))
     .     +((ycoor(i+1;n-i)-yi)*(ycoor(i+1;n-i)-yi))
     .     +((zcoor(i+1;n-i)-zi)*(zcoor(i+1;n-i)-zi))
c
c exclusions
c
        itemp2(1;nx)=inb(istart2;nx)
        itemp2(next;left)=1
        indexdyn(i;n-i+1)=q8vscatr(itemp2(1;n-i+1),itemp2(1;n-i+1);
     .  indexdyn(i;n-i+1))
        bitnotex(i+1;n-i)=indexdyn(i+1;n-i).eq.0
        indexdyn(i+1;n-i)=i
        iaddvec(i+1;n-i)=q8vgathr(iacivec(1;n),indexdyn(i+1;n-i);
     .  iaddvec(i+1;n-i))
        bitnot0(i+1;n-i)=iaddvec(i+1;n-i).ge.0.or.iac(i+1;n-i).ne.0
        ihbindex(i+1;n-i)=iaddvec(i+1;n-i)+iac(i+1;n-i)
        ihbarray(i+1;n-i)=q8vgathr(ico(1;900),ihbindex(i+1;n-i);
     .  ihbarray(i+1;n-i))
        bitnothb(i+1;n-i)=ihbarray(i+1;n-i).gt.0
c
c distance criterion
c
        bitdist(i+1;n-i)=distance(i+1;n-i).le.cut
        bitfinal(i+1;n-i)=bitdist(i+1;n-i).and.bitnotex(i+1;n-i)
        bitfinal(i+1;n-i)=(bitfinal(i+1;n-i).and.bitnothb(i+1;n-i))
        bitfinal(i+1;n-i)=bitfinal(i+1;n-i).and.bitnot0(i+1;n-i)
        itemp(i)=q8scnt(bitfinal(i+1;n-i))
        ihold=itemp(i)
        iar2(istart;ihold)=
     .  q8vcmprs(iindex(i+1;n-i),bitfinal(i+1;n-i);iar2(istart;ihold))
        istart=istart+ihold
        bitnothb(i+1;n-i)=.not.bitnothb(i+1;n-i)
        bitnothb(i+1;n-i)=bitnothb(i+1;n-i).and.bitnotex(i+1;n-i)
        bitnothb(i+1;n-i)=bitnothb(i+1;n-i).and.bitdist(i+1;n-i)
c
c h-bond list
c
        nhbi=q8scnt(bitnothb(i+1;n-i))
        jhb(nhbiter;nhbi)=q8vcmprs(iindex(i+1;n-i),bitnothb(i+1;n-i);
```

```
     .  jhb(nhbiter;nhbi))
        ihb(nhbiter;nhbi)=i
        ich(nhbiter;nhbi)=q8vcmprs(ihbarray(i+1;n-i),bitnothb(i+1;n-i);
     .  ich(nhbiter;nhbi))
        ich(nhbiter;nhbi)=vabs(ich(nhbiter;nhbi);ich(nhbiter;nhbi))
        nhbiter=nhbiter+nhbi
        istart2=istart2+nx
10       continue
        iar1(1;n)=itemp(1;n)
        nnb=istart-1
        nhb=nhbiter-1
        return
        end
```

## APPENDIX B: VECTORIZED NONBONDED ENERGY EVALUATION

To create vectors close to 3,500 the elements of IAR1 are sequentially summed until a value close to 3,500 is formed; this value is stored in the array ITEMP. The index of the last atom whose nonbonded partners contribute to this sum is stored in the array ITEMP2. For example, if the first two elements of ITEMP and ITEMP2 are:

| | | | |
|---|---|---|---|
| ITEMP: | 3498 | 3492 | .... |
| ITEMP2: | 98 | 205 | .... |

then in the first iteration of the outer Do loop of the nonbonded subroutine the vector lengths will be 3498 and all nonbonded interactions between the first 98 atoms and their nonbonded partners will be calculated; in the second iteration the vector lengths will be 3492 and all nonbonded interactions between atoms 99 through 205 and their nonbonded partners will be calculated.

The nonbonded energy of the system is then evaluated by looping over the total number of long vectors. To implement the block structure the indexing vector INDEXDYN is formed, containing the indices of atoms I repeated according to the number of nonbonded partners atom I has, viz., IAR1(I). For example, if atom 1 has four nonbonded partners and atom 2 has two nonbonded partners the first six elements of INDEXDYN are:

1,1,1,1,2,2,....

The program gathers the coordinates and parameter constants for the current iteration by using INDEXDYN to access the data for atom numbers I and IAR2 for the respective partners J.

The Lennard-Jones terms are computed first with the pairwise contributions being stored in the array ESUBI. This array is summed using the intrinsic function call to Q8SSUM and the result added to the accumulating sum ENB. Following the determination of ENB, the electrostatic energy, EEL, is evaluated for the atoms contained in the long vectors. The energy is summed and the entire procedure repeated NUMVEC times.

```
      subroutine nonbon (enb,eel)
      dimension xdist(3500),ydist(3500),zdist(3500)
      dimension iaddvec(3500),indexdyn(3500),cgi(3500),cgj(3500)
      dimension distance(3500),r12(3500),r6(3500),bitcut(3500)
      dimension xjcoor(3500),yjcoor(3500),zjcoor(3500)
      dimension xicoor(3500),yicoor(3500),zicoor(3500)
      dimension g(3500),xmult(3500),f2(3500),esubi(3500)
      dimension f1(3500),et1(3500),r2(3500),itemp(3500),itemp2(3500)
      dimension icnarray(3500),icnindex(3600)
      bit bitcut
c
      common/ /itemp,itemp2,indexdyn,cgi,cgj,xicoor,yicoor
     ..zicoor,xjcoor,yjcoor,zjcoor,xdist,ydist,zdist,distance
     .,r2,r6,r12,f1,f2,esubi,xmult,et1,g
```

```
c
      common /carts/ x(10500),xo(10500)
     .       /charge/ cg(3500)
     .       /cons1/ dielc,pnbfac,pelfac,ntypes,idist
     .       /info/ natom,ico(900),iac(3500)
     .       /nbterm/ cut,nbuck,ibio(3500),inb(15000)
     .       /parms/ rk(150),req(150),tk(250),teq(250),pk(150),
     .               phase(150),cn1(465),cn2(465),solty(30),pn(150)
     .       /select/ iar1(3500),iar2(208000)
c
      n=natom
      enb = 0.
      eel = 0.
      ehb = 0.
      dielc2=1./dielc
      one(1;3500)=1.0
      iione(1;3500)=1
      iacivec(1;n)=(iac(1;n)-iione(1;n))*ntypes
      iindex(1;3500)=q8vintl(1,1;iindex(1;3500))
      xcoor(1;n)=q8vgathp(x(1;0),3,n;xcoor(1;n))
      ycoor(1;n)=q8vgathp(x(2;0),3,n;ycoor(1;n))
      zcoor(1;n)=q8vgathp(x(3;0),3,n;zcoor(1;n))
c  set up block structure:
c  get the number of long vectors, NUMVEC
      ihold=0
      numvec=0
      iar1(n)=0
      i=0
450   continue
      i=i+1
       ihold=ihold+iar1(i)
      if(ihold.gt.3500) then
        numvec=numvec+1
        itemp2(numvec)=i-1
        itemp(numvec)=ihold-iar1(i)
        ihold=0
        i=i-1
        go to 450
      end if
      if(i.eq.n) then
      numvec=numvec+1
      itemp2(numvec)=i
      itemp(numvec)=ihold
      go to 451
     end if
      go to 450
451 continue
c -- end of block structure set up -- now do the calculation ------------
c istart  = the beginning position in itemp2
c istart2 = the beginning position of the next segment in the construction
c           of indexdyn
c istart3 = the starting position in iar2
c --- initialize counters --
      istart=1
      istart3=1
c loop through the total number of long vectors
      do 380 i=1,numvec
c create INDEXDYN
        istart2=1
        do 70 j=istart,itemp2(i)
          if(iar1(j).ne.0)then
            indexdyn(istart2;iar1(j))=j
            istart2=istart2+iar1(j)
          end if
70        continue
c nn = vector length
      nn=itemp(i)
c gather charges for electrostatic potential
      cgi(1;nn)=q8vgathr(cg(1;n),indexdyn(1;nn);cgi(1;nn))
      cgj(1;nn)=q8vgathr(cg(1;n),iar2(istart3;nn);cgj(1;nn))
c gather the constants for the Lennard-Jones potential
      iaddvec(1;nn)=q8vgathr(iacivec(1;n),indexdyn(1;nn);
     . iaddvec(1;nn))
      icnindex(1;nn)=q8vgathr(iac(1;n),iar2(istart3;nn);
     . icnindex(1;nn))
      icnindex(1;nn)=iaddvec(1;nn)+icnindex(1;nn)
      icnarray(1;nn)=q8vgathr(ico(1;900),icnindex(1;nn);
     . icnarray(1;nn))
      f1(1;nn)=q8vgathr(cn1(1;465),icnarray(1;nn);f1(1;nn))
      f2(1;nn)=q8vgathr(cn2(1;465),icnarray(1;nn);f2(1;nn))
```

```
c gather coordinates for all i and j
      xicoor(1;nn)=q8vgathr(xcoor(1;n),indexdyn(1;nn);
     . xicoor(1;nn))
      yicoor(1;nn)=q8vgathr(ycoor(1;n),indexdyn(1;nn);
     . yicoor(1;nn))
      zicoor(1;nn)=q8vgathr(zcoor(1;n),indexdyn(1;nn);
     . zicoor(1;nn))
      xjcoor(1;nn)=q8vgathr(xcoor(1;n),iar2(istart3;nn);
     . xjcoor(1;nn))
      yjcoor(1;nn)=q8vgathr(ycoor(1;n),iar2(istart3;nn);
     . yjcoor(1;nn))
      zjcoor(1;nn)=q8vgathr(zcoor(1;n),iar2(istart3;nn);
     . zjcoor(1;nn))
c compute distance between all i and j
      xdist(1;nn)=xicoor(1;nn)-xjcoor(1;nn)
      ydist(1;nn)=yicoor(1;nn)-yjcoor(1;nn)
      zdist(1;nn)=zicoor(1;nn)-zjcoor(1;nn)
      distance(1;nn)=xdist(1;nn)*xdist(1;nn)
     .              +ydist(1;nn)*ydist(1;nn)
     .              +zdist(1;nn)*zdist(1;nn)
c cutoff criterion -- xmult = 0.0 for distances .gt. cut
      bitcut(1;nn)=distance(1;nn).gt.cut
      where(bitcut(1;nn))
           xmult(1;nn)=0.0
      otherwise
           xmult(1;nn)=1.0
      end where
c get 1/r**2 for electrostatics; 1/r**6 and 1/r**12 for Lennard-Jones
      r2(1;nn)=1.0/distance(1;nn)
      r6(1;nn)=r2(1;nn)*r2(1;nn)*r2(1;nn)
      r12(1;nn)=r6(1;nn)*r6(1;nn)
      f1(1;nn)=f1(1;nn)*r12(1;nn)
      f2(1;nn)=f2(1;nn)*r6(1;nn)
      et1(1;nn)=f1(1;nn)-f2(1;nn)
      esubi(1;nn)=et1(1;nn)*xmult(1;nn)
c sum the individual contributions and add to total nonbonded energy
      enb=enb+q8ssum(esubi(1;nn))
c now do the electrostatic energy
      g(1;nn)=cgi(1;nn)*cgj(1;nn)*r2(1;nn)*dielc2
      esubi(1;nn)=g(1;nn)*xmult(1;nn)
c sum the component electrostatic terms and add to the system total
      eel=eel+q8ssum(esubi(1;nn))
c increment counters for the next formation of INDEXDYN
        istart=itemp2(i)+1
        istart3=istart3+itemp(i)
380   continue
      return
      end
```