A NEW SHARED-MEMORY PROGRAMMING PARADIGM FOR MOLECULAR DYNAMICS SIMULATIONS ON THE INTEL PARAGON

E.F. D’Azevedo
C.H. Romine

Mathematical Sciences Section
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

Date Published: December 1994

Research supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED
DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
# Contents

1 Introduction ............................................. 1
   1.1 DOLIB (Distributed Object Library) ................. 1
   1.2 SOTON_PAR ............................................ 2
2 Overview of Parallelization of SOTON_PAR with DOLIB .... 3
   2.1 Geometric Hashing ..................................... 3
   2.2 DOLIB implementation of movout ...................... 4
   2.3 Force evaluation ...................................... 6
   2.4 Dynamic Load Balancing ................................ 7
   2.5 Shifted Force Potential ............................... 8
3 Parallel Performance ...................................... 8
4 Appendix .................................................. 12
   4.1 subroutine movout .................................... 12
   4.2 subroutine adjustxyz .................................. 13
   4.3 subroutine hashxyz .................................... 15
   4.4 subroutine setxlist ................................... 19
   4.5 subroutine setuplist ................................... 22
   4.6 subroutine set4list ................................... 23
   4.7 subroutine copyxyz .................................... 25
   4.8 subroutine force ...................................... 29
   4.9 subroutine forceall ................................... 31
   4.10 subroutine forcecol ................................... 33
   4.11 subroutine forcecal ................................... 45
5 References ................................................ 51
List of Figures

2.1 Top view of columns used in processing column \((i, j)\). Only two new columns are needed in processing column \((i+1, j)\). . . . . . 7
A NEW SHARED-MEMORY PROGRAMMING PARADIGM FOR MOLECULAR DYNAMICS SIMULATIONS ON THE INTEL PARAGON

E.F. D’Azevedo
C.H. Romine

Abstract

This report describes the use of shared memory emulation with DOLIB (Distributed Object Library) to simplify parallel programming on the Intel Paragon. A molecular dynamics application is used as an example to illustrate the use of the DOLIB shared memory library. SOTON_PAR, a parallel molecular dynamics code with explicit message-passing using a Lennard-Jones 6-12 potential, is rewritten using DOLIB primitives. The resulting code has no explicit message primitives and resembles a serial code. The new code can perform dynamic load balancing and achieves better performance than the original parallel code with explicit message-passing.
1. Introduction

This report describes the use of shared memory emulation provided by DOLIB (Distributed Object Library) [3] for parallel programming of large-scale Molecular Dynamics (MD) codes on the Intel Paragon distributed memory supercomputer. SOTON_PAR [7], a parallel MD code that uses explicit message passing and Lennard-Jones atoms (6-12 potential), is rewritten using DOLIB shared memory primitives. The new MD prototype code can perform dynamic load balancing and achieves better performance than the original code that uses spatial decomposition. This report may also serve as a guide for scientists or programmers considering the use of DOLIB shared memory emulation in more sophisticated molecular dynamics simulations or particle-in-cell methods.

While this prototype code uses a simple Lennard-Jones 6-12 potential, more complicated force evaluation (such as including angular (three-body) forces for silicon and torsional (four-body) forces for organic polymers or proteins) can easily be added using the DOLIB shared memory framework. DOLIB’s dynamic load balancing capability is especially attractive for more demanding MD simulations.

1.1. DOLIB (Distributed Object Library)

DOLIB (Distributed Object Library) is a set of FORTRAN and C callable routines to emulate shared memory on distributed-memory environments such as Intel multiprocessors and PVM clusters of workstations.

DOLIB supports runtime dynamic creation and destruction of one-dimensional arrays. Explicit gather and scatter operations provide access to array elements. DOLIB is portable since no language extension is introduced and no preprocessor, compiler or operating system support is required. DOLIB provides an atomic accumulate operation \texttt{axpby} \( y(index(:) \leftarrow \alpha \times x(:) + \beta \times y(index(:)) \), \) where \( x \) is a local array, \( y \) is a globally shared array, and \( index \) is an array of indices for \( y \). \texttt{axpby} is intended for use in finite element matrix assembly, with no need for explicit lock/unlock. Note that \( \alpha = 1, \beta = 0 \) corresponds to a scatter operation. DOLIB also provides automatic caching of read-only data to reduce
message traffic. However, the programmer has the responsibility of flushing the cache to maintain coherency during concurrent updates.

A global array in DOLIB is stored as fixed-size pages in a block wrapped mapped fashion across all processors. DOLIB translates requests for remote data (gather) or update (scatter) into the appropriate message sent to the "owner" processor of that data page. These DOLIB requests are then serviced by the IPX [6] message system.

DOLIB supports a generalized atomic update operation ($axpbyz$),

\[
\begin{align*}
z(:) &\leftarrow y(index(:)), \\
y(index(:)) &\leftarrow \alpha * x(:) + \beta * y(index(:)) \\
x(:) &\leftarrow z(:)
\end{align*}
\]

Operation $axpbyz$ can also be used to implement a vector of counting semaphores.

1.2. SOTON_PAR

The original version of SOTON_PAR\(^1\) MD code ran on the Intel iPsc/2. The version we obtained was substantially modified by David Walker to run efficiently on Intel Paragon multiprocessors. SOTON_PAR models short range inter-atomic interactions by using a link-cell (geometric hashing) algorithm where all particles are hashed into three-dimensional $N_b \times N_b \times N_b$ bins. The bin size is at least the cut-off distance ($r_c$) used in the short-range force evaluation. SOTON_PAR exploits the symmetry of Newton's Third Law by examining atoms in only 13 (instead of 26) neighboring cells. It uses a spatial decomposition [8] and requires message exchanges with neighbor processors at each time step as atoms migrate across processor boundaries. Many of the MD algorithms used in SOTON_PAR are described in the book *Computer Simulation of Liquids* [1]. The code also uses a "shifted-force" [9] Lennard-Jones 6-12 potential to avoid a discontinuity at the

\[^{1}\text{The source code for SOTON_PAR is available from the CCP5 archive at ftp://ftp.dl.ac.uk/ccp5/SOTON_PAR/mdplli3cu.master}\]
cut-off distance.

The code consists of two parts: the "host" code runs on the service partition and controls the I/O, allocation and loading of nodes etc.; the "worker" code runs on the compute nodes and performs most of the work in geometric hashing and force evaluations.

The code assumes an \( N_c \times N_c \times N_c \) FCC (Face Centered Cubic) lattice scaled to the canonical unit cube \([-0.5,0.5] \times [-0.5,0.5] \times [-0.5,0.5]\), with a total of \( N = 4N_c^3 \) atoms. A periodic boundary condition is imposed on all sides. A simple Verlet leap-frog time stepping scheme is used to update the particle positions.

2. Overview of Parallelization of SOTON_PAR with DOLIB

We have written a new parallel MD prototype code based on SOTON_PAR using DOLIB shared memory primitives. The new code shares much of the overall structure of SOTON_PAR with most of the host code for performing I/O executed by node 0. We believe the new code is much easier to write and understand without the complexities of explicit message passing code. We have also added several enhancements: (1) initialize with Boltzmann distribution for faster convergence to equilibrium, (2) a more robust parallel cold-start mechanism even for very large configurations, (3) an option for dynamic load balancing, (4) an option for using double precision for force computation.

We shall discuss two of the most time consuming computational kernels in SOTON_PAR: movout, which performs geometric hashing, and force, which performs the force evaluations.

2.1. Geometric Hashing

The link-cell method [4] is commonly used to speed up MD force calculations by minimizing the number of neighboring atoms that must be checked for possible interaction (based on a cut-off distance of \( r_c \), beyond which particle interaction is assumed to be negligible). At each time step, all the atoms are hashed into
three-dimensional bins or cells of side length at least $r_c$. Each particle can thus interact only with atoms within the same bin or the 26 surrounding bins. Hashing the atoms can be easily parallelized and requires only $O(N)$ work. Only 13 of the neighboring bins need to be examined if Newton’s Third Law ($f_{ij} = -f_{ji}$) is applied. The link-cell method is memory efficient, requiring only $O(N^3)$ storage for the 3-D bins, 9 real vectors of length $N$ ($x, y, z, v_x, v_y, v_z, f_x, f_y, f_z$ for the positions, velocities and forces, respectively) and an integer vector of length $N$ for storing the linked list.

Another common technique is to construct and maintain for each atom, a list of neighboring atoms [10]. This list is updated every few time steps. The advantage provided by the neighbor list is that once the list is built, examining it for possible interaction is much quicker than checking all other atoms in the neighboring bins. However, the neighbor list for each atom commonly grows quite long (70 or more atoms per list), and the resulting high cost in memory is prohibitive for very large scale MD calculations.

SOTON_PAR is based on a spatial decomposition and uses the link-cell method. Each processor is responsible for a spatial region and the corresponding subset of 3-D bins. SOTON_PAR uses message passing code to exchange atoms with neighbor processors as atoms migrate across processor boundaries. A highly non-uniform distribution of particles may result in a serious load-imbalance.

2.2. DOLIB implementation of movout

The link-cell algorithm in SOTON_PAR constructs a linked list of atoms for each bin. We implement the link-cell method by performing a reordering or renumbering so that all atoms in a bin are contiguously numbered, e.g., if each bin has 10 atoms, then after the reordering, atoms 1 to 10 are contained in the first bin, atoms 11 to 20 are contained in the second bin, etc. The reordering permits the use of the most efficient DOLIB contiguous block gather/scatter operations on long vectors.

Two passes are required for the geometric hashing and reordering. The first
pass performs the geometric hashing and stores the result of the mapping. For each bin, we compute the number of atoms to be assigned. We then set up and allocate storage for each bin using a pointer array \texttt{xlist}. The second pass then performs the actual reordering and data movement. Note that vectors \((f_x, f_y, f_z)\) are used as temporary storage and will be cleared again for force computation.

The geometric hashing and reordering is performed in several routines:

\texttt{adjustxyz}: After a time step, particles close to the boundary might exit the canonical box \([-0.5, 0.5] \times [-0.5, 0.5] \times [-0.5, 0.5]\). We implement a periodic boundary condition by adjusting the \((x, y, z)\) coordinate to reintroduce the particle at the opposite face. Hence,

\[
\begin{align*}
\text{if } (x > 0.5) & \quad x = x - 1.0 \\
\text{if } (x < -0.5) & \quad x = x + 1.0
\end{align*}
\]

\texttt{hashxyz}: Geometric hashing is performed as a simple divide and integer truncation operation. An atom with coordinates \((x, y, z)\) will be hashed to 3-D bin \((i, j, k)\) by

\[
\begin{align*}
i & = 1 + \text{int}((x - (-0.5))/r_c) \\
j & = 1 + \text{int}((y - (-0.5))/r_c) \\
k & = 1 + \text{int}((z - (-0.5))/r_c)
\end{align*}
\]

The result of the hashing is stored in array \texttt{imap} for use in the second pass. The array \texttt{imap} requires \(N_b^3\) words of memory.

\texttt{setxlist}: We use a pointer array \texttt{xlist} to mark for each bin, the beginning position (or label) of each contiguous block of atoms assigned to the bin. Let \(m_i\) be the number of particles assigned to the \(i^{th}\) bin, then \(xlist(1) = 1, xlist(2) = 1 + m_1, xlist(3) = xlist(2) + m_2, \ldots, xlist(k+1) = xlist(k) + m_k\), etc. The array \texttt{xlist} requires \(N_b^3\) words of memory.
setuplist: We shall use xlist as pointer to the next available storage position for each bin. Thus as we assign particles into bins, xlist has to be incremented in parallel with an atomic operation. We avoid the use of inefficient lock/unlock by using the DOLIB apbyz operation.

copyxyz: Once the assignment mapping is determined, vectors \((x, y, z)\) and \((v_x, v_y, v_z)\) are permuted using \((f_x, f_y, f_z)\) as temporary storage. Vectors \((f_x, f_y, f_z)\) are then cleared again before the force computation. The arrays \((x, y, z)\), \((v_x, v_y, v_z)\) and \((f_x, f_y, f_z)\) require \(9N\) words of memory for \(N\) atoms. If the forces \((f_x, f_y, f_z)\) are to be computed in double precision, \(12N\) words are required.

adjustxlist: Finally we reset xlist to point to the beginning position of each contiguous block.

Note that as the MD system converges to an equilibrium state, most of the particles will be hashed into the same bins to which they belonged in the previous time step, thus requiring only a small amount of data movement.

In Section 3 we list the proportion of time spent in each routine.

2.3. Force evaluation

We take advantage of Newton’s 3rd Law \((f_{ij} = -f_{ji})\) to compute a force interaction only once for each atom pair. This means only 13 (instead of 26) neighboring bins must be examined. Lomdahl [5] describes the selection of 13 neighbors as a particular “3D interactions path”, though in fact any stencil of 13 neighbors can be chosen, if consistently used.

For simplicity, we consider a two-dimensional partition of the computation. Thus, we group all force interactions for each column of \(N_b\) cells of the 3D mesh of bins as one unit. The \(N_b \times N_b\) columns are then block partitioned and assigned to individual processors. Because all atoms in each bin are contiguously ordered (see Section 2.2), DOLIB’s efficient contiguous block gather/scatters can be used. Moreover, we exploit data reuse by selecting the 13 neighboring cells to be
2.3. Further Communication Reduction

in 5 neighboring columns (see Figure 2.1). To process column \((i, j)\) we require data from neighboring columns \((i-1, j+1), (i, j+1), (i+1, j), (i+1, j+1)\); to process the next column \((i+1, j)\), data in columns \((i+1, j), (i, j+1), (i+1, j+1)\) can be reused. Only data for two new columns \((i+2, j)\) and \((i+2, j+1)\) must be brought in. This organization reduces the amount of communication required for gather/scatter operations.

Another advantage of this reorganization is that force evaluation in each of the bins requires no further communication. Automatic thread parallelization by the Intel MP node Paragon compiler is thus simplified.

2.4. Dynamic Load Balancing

We include an option to estimate a work measure for each column, using this to distribute columns to processors to avoid load imbalance. One simple measure of work load is the total number of atoms in the column. For a uniform distribution of atoms in the domain, this is generally sufficient to attain good load-balance. However, for nonuniform distributions, a more reasonable work measure for each
bin is to count the total number of possible atom-atom interactions among other particles in the 13 neighboring bins. Both of these work estimates are provided as options to the load-balancing routine.

The load balancing strategy then computes the overall total and average amount of work for each processor, assigning columns to processors to satisfy this average using a greedy algorithm. Future versions will provide a more sophisticated bin-packing algorithm. However, for the simple Lennard-Jones fluid simulations, each bin has approximately the same number of atoms and induces almost no load imbalance.

2.5. Shifted Force Potential

SOTON_PAR implements a shifted force Lennard-Jones 6-12 potential [1, page 145] where a small linear term is added to the potential so that it is continuous and its derivative is zero at the cutoff distance. The code also computes the virial and potential energy along with force calculations. Each atom-pair takes 8 flops and 1 compare to determine whether they are within the cut-off distance. If they are, then another 29 flops, 2 divides and 1 square root operation are further required. In our experience, the accumulation into total potential energy may encounter catastrophic cancellation that can yield as few as 4 significant digits of precision. We have implemented an option using separate double precision variables \( v_{\text{pos}} \) and \( v_{\text{neg}} \) for accumulating positive and negative potential values, which reduces the amount of numerical cancellation and produces more consistent results as the number of processors is varied.

3. Parallel Performance

Our parallel codes were tested on an Intel MP node Paragon system. Each MP node contains 3 CPUs and at least 64MBytes of memory in a local shared memory configuration. By default, one CPU is a dedicated message co-processor. The second CPU runs the main computational thread. The third CPU can be utilized by
automatic thread parallelization by the Paragon MP Fortran compiler. The MP node can also be configured in “turbo” mode where all three CPUs are dedicated for computation but with some degradation in communication performance. All our MD computations were done in single precision in non-turbo mode, although we have also implemented a double precision option for force computations.

We tested our code with a benchmark problem described in Plimpton [8, page 23] for a Lennard-Jones 6-12 potential with reduced density $\rho = 0.8442$, and reduced temperature $T = 0.72$. The system is initialized with a fcc lattice and randomized velocities chosen from a Boltzmann distribution. The integration time step is $0.00462$ in reduced units, and cut-off distance is $r_c = 2.5\sigma$. Plimpton estimates there are about 55 neighbors interacting with each atom at every time step (out of about 177 atom-pairs examined). If we count a divide as 5 flops and a square root also as 5 flops, then (from Section 2.5) each atom requires approximately $9 \times 177 + 44 \times 55 = 4013$ flops for force evaluation. Updating the velocities, positions and accumulation of total kinetic energy in movea and moveb requires another 30 flops per atom.

Table 3.1 shows averaged run times per time step for a 500,000 atom simulation ($50 \times 50 \times 50$ lattice). Initialization and setup time are excluded. We achieved faster run times without the overhead of computing work measure and dynamic load balancing, since the problem is already well-balanced. The serial run was performed using the same “parallel” code on a single processor. Note that no messages were generated in the DOLIB gather/scatter since on a single processor these are translated into memory copies.

For the 500,000 atom problem, our code achieved a speed of 0.175ms/atom per time step on a single processor (with parallel threads) with no messages generated. About 75% of the overall time (per time step) was spent in force computation and about 20% in performing reordering in movout. Within movout, about 42% of time was spent in performing 3-D bin hashing (first pass) in hashxyz, about 17% in allocating storage with DOLIB axpbyz operation in routine setuplist and about 35% in performing (second pass) actual reordering of vectors ($x, y, z,$
Table 3.1: Averaged runtimes (in sec) per timestep for 500,000 nodes problem.

<table>
<thead>
<tr>
<th>Processor</th>
<th>1 thread</th>
<th>2 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>137.5</td>
<td>95.4</td>
</tr>
<tr>
<td>2</td>
<td>73.1</td>
<td>51.8</td>
</tr>
<tr>
<td>4</td>
<td>37.6</td>
<td>27.2</td>
</tr>
<tr>
<td>8</td>
<td>19.7</td>
<td>13.2</td>
</tr>
<tr>
<td>16</td>
<td>10.3</td>
<td>7.1</td>
</tr>
<tr>
<td>32</td>
<td>5.6</td>
<td>4.3</td>
</tr>
<tr>
<td>64</td>
<td>3.3</td>
<td>2.4</td>
</tr>
</tbody>
</table>

$v_x, v_y, v_z$ in routine copyxyz. On 64 processors with parallel threads, the speed decreased to about 0.31 ms/atom per processor per time step. About 60% was in force and 30% in movout. Within movout, about 27% was in hashxyz, 21% in setuplist and about 38% was in copyxyz. Note however, there is message passing and servicing of remote memory requests even within the force computations.

By comparison, the original SOTON_PAR host/node codes required about 17.1 sec on 8 processors for a 256,000 atom (40 x 40 x 40 lattice) with no parallel threads. This is about 0.534 ms/atom/processor. Our code (with no parallel threads) takes about 10.4 sec per time step. This is about 0.333 ms/atom. With parallel threads enabled, the average time reduced to 7.7 sec per step, or 0.246 ms/atom/processor.

On a much larger problem with 32 million atoms (200 x 200 x 200 lattice) our code takes about 205.3 sec on 32 processors and 103.7 sec on 64 processors, which is about 0.207 ms/atom per processor. This yields an approximate overall rate (including message passing overhead) of 19.5 MFlop/sec for each node (with parallel threading enabled). Although we used single precision in our codes, our run times compare quite favorably with the times (performed in double precision) of 2.03 ms/atom/processor on the Intel Delta [8] and 1 ms/atom/processor [2] on a 1024 processor CM-5 both using a spatial decomposition and linked-cell method.
Acknowledgments

The authors take this opportunity to express appreciation to Bob Marr, Ron Peierls and Joe Pasciak for the IPX package, which simplified the development of DOLIB. We also thank Bill Shelton and David Walker for their insight and advice on Molecular Dynamics Simulations.
4. Appendix

In this appendix, we list the RATFOR source code for some of the key routines used in movout permutation and force computations.

4.1. subroutine movout

```ratfor
#include "stdinc.h"
/* routine to setup hash table and other data structures for force calculations */

subroutine movout()
#include "globals.h"
#include "parallel.h"

if (useminspace)
   /* disable everything */
   call dodisable(0);
   call dogsync0;
   call dodestroy(IGfxyz);
   call dogsync0;
   name = 'imap(natoms)' // char(0);
   ctype = 'integer' // char(0);
   gsize = natoms; psize = pagesize; blocksize = 1;
   call dodeclare(IGimap, name, gsize, ctype, psize, blocksize);
   call dogsync();

#endif
```

/* make sure all xyz coordinates are within canonical box */
call adjustxyz();

/* first pass in hashing to count number of atoms in each bin cell */
call hashxyz();

/* setup up pointer list */
call setxlist();

/* setup up numlist */
call setuplist();

/* second pass in hashing to copy xyz coordinates into the 'permuted' array */
call copyxyz();

/* reset xlist */
call adjustxlist();

#ifdef USE_PROFILE
    call profend('movout');
#endif
    return;
[end

4.2. subroutine adjustxyz

#include "stdinc.h"

/* perform adjustment of xyz to be within canonical box of
   [xmin,xmax] x [ymin,ymax] x [zmin,zmax] */
#define logdev 16

subroutine adjustxyz()
    #include "globals.h"
    #include "parallel.h"
{
    intrinsic int,max,min,abs,sqrt;
    integer ipage,npages,ipagesstrt;
    integer ni,i,istrt,iend,isize;

    /* simulation in box [-0.5,0.5] */
    real xmin,xmax,xadjust;
    real ymin,ymax,yadjust;
    real zmin,zmax,zadjust;
parameter(xmin=-0.5,xmax=0.5,xadjust=(xmax-xmin));
parameter(ymin=-0.5,ymax=0.5,yadjust=(ymax-ymin));
parameter(zmin=-0.5,zmax=0.5,zadjust=(zmax-zmin));

real one,zero;
parameter(one=1.0,zero=0.0);

real xyz(3,pagesize);

#define rx(i) xyz(1,i)
#define ry(i) xyz(2,i)
#define rz(i) xyz(3,i)

logical isxok,isyok,iszok, isallok, isformatted;

/* start execution */
#ifif USE_PROFILE
call profstart('adjustxyz');
#endif
call dogsync();

npages = int(natoms/pagesize) + 1;
ipagestrt=myid;
call dodisable(IGxyz);

/* scatter operation */
call openrbuf(IGxyz, one, zero);

doloop4(ipage,ipagestrt,npages,nproc) {
    istrt = 1 + ipage*pagesize;
    iend = min(natoms, istrt+pagesize-1),
    isize = iend - istrt + 1;
    if (isize <= 0) { break; }

    call dowait(dobrgather(IGxyz, 3*isize, idxyzz(1,istrt), xyz));

    if (IDEBUG >= 3) {
        write(logdev,*) 'adjustxyz: xyz before update ',
        write(logdev,9001) (istrt+(i-1),rx(i),ry(i),rz(i),i=1,isize);
        format((ix,i6,'.',3(1x,lpe14.5)));
    }

    doloop(i,1,isize) {

        isallok = TRUE;
        if (rx(i) < xmin) { rx(i) = rx(i) + xadjust; isallok = FALSE;};
        if (rx(i) > xmax) { rx(i) = rx(i) - xadjust; isallok = FALSE;};
        if (ry(i) < ymin) { ry(i) = ry(i) + yadjust; isallok = FALSE;};
        if (ry(i) > ymax) { ry(i) = ry(i) - yadjust; isallok = FALSE;};
        if (rz(i) < zmin) { rz(i) = rz(i) + zadjust; isallok = FALSE;};
        if (rz(i) > zmax) { rz(i) = rz(i) - zadjust; isallok = FALSE;};
    }
}

}
if (isallok) { next; }

/* -------------- */

ni = istrt + (i-1);
if (IDEBUG >= 2) {
    isxok = (xmin <= rx(i)) & (rx(i) <= xmax);
    isyok = (ymin <= ry(i)) & (ry(i) <= ymax);
    iszok = (zmin <= rz(i)) & (rz(i) <= zmax);
    isallok = isxok & isyok & iszok;

    if (!isallok) {
        write(*,*) ' ** adjustxyz: invalid xyz after adjustment';
        write(*,*) 'ni,rx,i,ry,i,rz,i,rxi,ryi,rzi',ni,rx(i),ry(i),rz(i);
        isformatted = TRUE;
        call outconf( isformatted ); /* formatted dump */
    }

    ASSERT(isallok,
          ' ** adjustxyz: invalid xyz after adjustment',ni);
    } /* if (DEBUG >= 2) */

    /* buffer the update */
    call storebrbuf( IGxyz,3, idxxyz(i,ni) , xyz(i,i) );
}; /* end do i */

}; /* end do ipage */

call closerbuf(IGxyz);
call dogsync();

#ifdef USE_PROFILE
    call profend('adjustxyz');
#endif

return;
} end

4.3. subroutine hashxyz

#include "stdinc.h"

/* perform hashing of xyz to count number of atoms in each bin */

#define USE_WCACHE 1
subroutine hashxyz()
#include "globals.h"
#include "parallel.h"
{
    intrinsic int,max,min,abs,sqrt;
    integer ipage,npages,ipagestrtr;
    integer i,istrt,iend,isempty;
    integer ix,iy,iz;
    integer ival,gsize,psize;

    integer ilist(pagesize), mcount;

    logical usedivide;
    ifdef USE_DIVIDE
        parameter(usedivide=TRUE);
    else
        parameter(usedivide=FALSE);
    endif

    logical usewcache;
    ifdef USE_WCACHE
        parameter(usewcache=TRUE);
    else
        parameter(usewcache=FALSE);
    endif

    logical usescallsort;
    ifdef USE_SORT
        parameter(usecallsort=TRUE);
    else
        parameter(usecallsort=FALSE);
    endif

    integer ibin,icount;
    logical issame;
    /* simulation in box [-0.5,0.5] */

    real xmin,xmax,xadjust;
    real ymin,ymax,yadjust;
    real zmin,zmax,zadjust;
    parameter(xmin=-0.5,xmax=0.5,xadjust=(xmax-xmin));
    parameter(ymin=-0.5,ymax=0.5,yadjust=(ymax-ymin));
    parameter(zmin=-0.5,zmax=0.5,zadjust=(zmax-zmin));

    real rxi,ryi,rzi, rcutinv;
    real xyz(3,pagesize);
    #define rx(i) xyz(1,i)
    #define ry(i) xyz(2,i)
    #define rz(i) xyz(3,i)

    integer list(pagesize), ivalue(pagesize);
    /* =========== start execution =========== */
ifdef USE_PROFILE
    call profstart('hashxyz');
endif

call doenable( IGxyz );
call dodisable( IGncount );

/* zero out global array ncount(); */
ival = 0;
gsize = nbinx*nbiny*nbinz; pagesize = pagesize;
call givecfill( IGncount, ival, gsize, pagesize );
if (!useucache) {
call ifill( pagesize, ival, 1 );
}

if (!usedivide) {  
rcutinv = 1.0/rcut;
}

npages = int( natoms/pagesize ) + 1;
ipagestrt=myid;
if (useucache) { call openibuf( IGncount, 1, 1 ); }
doloop(ipage,ipagestrt,npages,mproc) {  
    iend = min(natoms, istrt+pagesize-1);
    isize = iend - istrt + 1;
    if (isize <= 0) { break; }

call dowait(dobrgather( IGxyz, 3*isize, 
        idxyi(i,istrt), xyz ));

doloop(i,l,isize) {

    if (usedivide) {  
        ix = i+int((rx(i) - xmin)/rcut );
        iy = i+int((ry(i) - ymin)/rcut );
        iz = i+int((rz(i) - zmin)/rcut );
    } else {  
        ix = i+int((rx(i) - xmin)*rcutinv );
        iy = i+int((ry(i) - ymin)*rcutinv );
        iz = i+int((rz(i) - zmin)*rcutinv );
    }

    if (IDEBUG >= 2) {  
        if (ix < 1) { ix = ix + nbinx; }
        if (ix > nbinx) { ix = ix - nbinx; }

        if (iy < 1) { iy = iy + nbiny; }
        if (iy > nbiny) { iy = iy - nbiny; }

        if (iz < 1) { iz = iz + nbiny; }
        if (iz > nbiny) { iz = iz - nbiny; }
    }
if (iz < 1) { iz = iz + nbinz; }
if (iz > nbinz) { iz = iz - nbinz; }

list(i) = idxncount(ix,iy,iz);
} /* end do i */

/* save hash results into imap */
call dobiscatter (IGimap, isize, idximap(istrt), list);

/* --- accumulated into hash table */

if (useucache) {

    /* attempt compression, instead of adding ncount(:) by icount times,
       simply add by icount only once */
    if (usesort) {
        call shell( list, isize);
    }

    mcount = 0;
    ibin = -1; icount = 0;
    doloop(i,l,isize) {  
        issame = (ibin == list(i));
        if (issame) { icount = icount + 1; }
        else {
            if (icount > 0) { /* flush out */
                /* call storeibuf( IGncount, 1, ibin, icount ); */
                mcount = mcount + 1;
                ilist(mcount) = ibin;
                ivalue(mcount) = icount;
            }
            ibin = list(i); icount = 1;
        }
    } /* end doloop */

    if (icount > 0) { /* flush out last remaining part */
        /* call storeibuf( IGncount, 1, ibin, icount ); */
        mcount = mcount + 1;
        ilist(mcount) = ibin;
        ivalue(mcount) = icount;
    }

    if (mcount > 0) {
        call storeibuf( IGncount, mcount, ilist, ivalue );
    }
    else {

        /* by pass intermediate buffering */
        call doiapby( IGncount, isize, list, ivalue, 1,1 );
    }
}
if (usewcache) { call closeibuf(INcount); }

call dogsync();

#ifdef USE_PROFILE
    call profend('hashxyz');
#endif

return;

end

4.4. subroutine setxlist

#include "stdinc.h"

#define logdev 16

/* given the ncount global array is complete, compute the xlist pointer array */

subroutine setxlist()
#include "globals.h"
#include "parallel.h"

intrinsic max, min;

#define ndim (8*pagesize)

integer istrt, iend, isize;
integer i, ip, nbins;
integer ncount(ndim), xlist(ndim);

#define maxproc (2*1024)

integer ipstart(maxproc+1);
integer mycount(maxproc), icount;

integer iproc, ibstrt, ibend, ibsize;

/* for simplicity, let processor 0 handle it, may not scale well for very large problems */

/* ========== start execution ========== */

#ifdef USE_PROFILE
    call profstart('setxlist')
#endif

#define
/* each processor compute part of the total count */

nbins = nbinx*nbiny*nbinz;
call partition( nbins, nproc, myid, ibstrt, ibend);
ibsize = ibend - ibstrt + 1;
doloop(iproc,1,nproc) { mycount(iproc) = 0; }
call doenable( IGncount );
iproc = myid + 1;
doloop4(istrt,ibstrt,ibend,ndim) {
    iend = min(ibend, istrt+ndim-1);
    isize = (iend - istrt + 1);
    if (isize <= 0) { break; }
    call dowait( dobigather( IGncount,isize,istrt,ncount ));
    icount = 0;
doloop(i,1,isize) {
        icount = icount + ncount(i);
    }
    mycount(iproc) = mycount(iproc) + icount;
    ASSERT( mycount(iproc) > 0,
        'setxlist: possible integer overflow ', mycount(iproc));
}; /* end do istrt */

call dogsync();
GISUH( mycount, nproc );

/* double check */
if (IDEBUG >= 2) {
    if (myid == 0) {
        write(logdev,'(s,18)') (mycount(iproc),iproc=1,nproc);
    }

    doloop(iproc,1,nproc) {
        call partition( nbins, nproc, iproc-1, ibstrt, ibend );
        icount = 0;
        doloop4(istrt,ibstrt,ibend,ndim) {
            iend = min(ibend, istrt+ndim-1);
            isize = (iend - istrt + 1);
            if (isize <= 0) { break; }
            call dowait( dobigather( IGncount,isize,istrt,ncount ));
            doloop(i,1,isize) {
                ASSERT( ncount(i) > 0,
                    'setxlist: ncount(i) <= 0',ncount(i) );
                icount = icount + ncount(i);
            }
        }
        ASSERT( icount == mycount(iproc),
            'setxlist: icount != mycount(iproc)',mycount(iproc));
    }; /* end do iproc */
}
/*
set up xlist so that
xlist(1) = 1,
xlist(2) = xlist(1) + ncount(1), ...
xlist(k+1) = xlist(k) + ncount(k)...
at beginning of call to copyxyz
*/

ipstart(1) = 1;
doloop(iproc, l, nproc) {
ipstart(iproc+1) = ipstart(iproc) + mycount(iproc);
}
call dodisable( IGxlist );
call partition( nbins, nproc, myid, ibstrt, ibend );

iproc = myid+1;
ip = ipstart(iproc);
doloop4(istrt, ibstrt, ibend, ndim) {
iend = min(ibend, istrt+ndim-1);
isspace = iend - istrt + 1;
if (isspace <= 0) { break; };

call dowait( dobigather( IGncount, isize, (istrt), ncount));
doloop(i, 1, isize) {
xlist(i) = ip;
ip = ip + ncount(i);
}
call dobiscatter( IGxlist, isize, istrt, xlist );
} /* end do istrt */
call dogsync();

if (DEBUG >= 2) {
if (myid == 0) {
ip = 1;
doloop4(istrt, 1, nbins, ndim) {
iend = min(nbins, istrt+ndim-1);
isspace = (iend - istrt + 1);
if (isspace <= 0) { break; };

call dowait( dobigather( IGncount, isize, istrt, ncount ));
call dowait( dobigather( IGxlist, isize, istrt, xlist ));
}

if (DEBUG >= 3) {
write(logdev,*) 'ncount(:) ';
}
write(logdev,9001) (ncount(i),istrt+(i-1),i=1,1isize);
write(logdev,*):'xlist(:)';
write(logdev,9001) (xlist(i),istrt+(i-1),i=1,1isize);
format(4( i6,'(',i6,')')));
}
doloo(i,1,1isize){
  if (xlist(i) i= ip) {
    write(logdev,*) 'ni, ip, xlist',
      (istrt+(i-1)),ip,xlist(i);
  }
  ip = ip + ncount(i);
}
}; /* end do istrt */
}; /* end if (myid) */
}; /* end if (DEBUG) */
call dogsync();

#ifdef USE_PROFILE
call profend('setxlist')
#endif
effect return;
}
end

4.5. subroutine setuplist

#include "stdinc.h"
#define SETLIST_OPTION 4

subroutine setuplist()
#include "globals.h"
#include "parallel.hl"
C
if (SETLIST_OPTION == 1) {
  /* use long vector gathers */
call set1ist();
}
else if (SETLIST_OPTION == 2) {
  /* use short vector gathers, take advantage of pagesize */
call set2list();
}
else if (SETLIST_OPTION == 3) {

4.6. subroutine set4list

#include "stdinc.h"

/* use a long vector gather */

subroutine set4list()
#include "globals.h"
#include "parallel.h"
{
    integer ndim;
    parameter(ndim=8*(pagesize));
    integer list(ndim),ipos(ndim),numlist(ndim);
    integer istrt,iend,isize;
    integer ibstart,ibend,ibsize, gsize;
    integer maxmcount; parameter(maxmcount=ndim);
    integer icount,mcount,ibis, ni,ip, i,j;
    integer binlist(maxmcount), countlist(maxmcount);
    integer idest(ndim), nilist(ndim);
    logical isame;
    /* save storage */
    equivalence (nilist,binlist);
    /* ========= start execution ========= */
#define USE_PROFILE
    call profstart('set4list');
#undefif
    gsize = natoms;
    call dodisable( IGxlist );
call partition( gsize, nproc, myid, ibstart, ibend );

dollop4( istrt, ibstart, ibend, ndim) {
    iend = min(ibend, istrt + ndim-1);
    isize = (iend - istrt + 1);
    if (isize <= 0) { break; }

call dowait(dobigather( IGimap, isize, idximap(istrt), list ));

    /* attempt compression, instead of adding one k times,
       try to add k only once */
    mcount = 0; icount = 0; ibin = -1;
    doloop(1,1,isize) {
        issame = (ibin == list(i));
        if (issame) { icount = icount + 1; }
        else {
            if (icount > 0) { /* store these updates */
                mcount = mcount + 1;
                ASSERT( mcount <= maxmcount,
                    ' * set4list: mcount > maxmcount ', mcount );
                binlist(mcount) = ibin;
                countlist(mcount) = icount ;
            }
            ibin = list(i); icount = 1;
        }
    }

    /* handle last remaining case */
    if (icount > 0) { /* store these updates */
        mcount = mcount + 1;
        ASSERT( mcount <= maxmcount,
            ' * set4list: mcount > maxmcount ', mcount );
        binlist(mcount) = ibin;
        countlist(mcount) = icount ;
    }

    /* set up ipos(:) for axpbyz operation */
    ASSERT( mcount >= 1,
        ' * set4list: invalid mcount ', mcount );

    /* add by how much, note axpbyz operation overwrites ipos(:) */
    call icopyc mcount, countlist, 1, ipos, 1;

call doiaxpbyz( IGxlist, mcount, binlist, ipos, 1,1 );

    /* prepare for scatter */
    ni = istrt; ip = 1;
    doloop(1,1,mcount) {
        doloop(j,1,countlist(i)) {
            idest(ip) = (ipos(i)+1) + j;
            nilist(ip) = ni;
            ni = ni + 1;
            ip = ip + 1;
        }
4.7. subroutine copyxyz

#include "stdinc.h"
#define logdev 16
/* perform hashing of xyz and copy coordinates to new array */
#define ISWAP( ia, ib ) { item = ia; ia = ib; ib = item; }

subroutine copyxyz()
#include "globals.h"
#include "parallel.h"
{

  logical usedfxyz;
  ifndef USE_DEFXYZ
    parameter(usedfxyz=TRUE);
  else
    parameter(usedfxyz=FALSE);
  endif

  STRING name; STRING ctype;
  integer psize, gsize, blocksize;

  integer ibstart, ibend;

intrinsic int, max, min, abs, sqrt;

integer ipage, npages, ipagelstr;
integer i, istrt, iend, isize, ip, jp, itemp, ni;
integer ix, iy, iz;

;/* simulation in box [-0.5,0.5] */
real xmin, xmax, xadjust;
real ymin, ymax, yadjust;
real zmin, zmax, zadjust;

parameter (xmin=-0.5, xmax=0.5, xadjust=(xmax-xmin));
parameter (ymin=-0.5, ymax=0.5, yadjust=(ymax-ymin));
parameter (zmin=-0.5, zmax=0.5, zadjust=(zmax-zmin));

logical useminspace;
#ifdef USE_MINSPACE
parameter(useminspace=TRUE);
#else
parameter(useminspace=FALSE);
#endif

#define ndim pagesize

integer numlist(ndim);
integer iposxyz(3,ndim);
real xyz(3,ndim); real vxyz(3,ndim);
equivalence (xyz, vxyz);
#define rx(i) xyz(l,i)
#define ry(i) xyz(2,i)
#define rz(i) xyz(3,i)

;/* ========= start execution ======== */
#ifdef USE_PROFILE
call profstart('copyxyz')
#endif

;/* disable everything */
call dodisable( 0 );

if (useminspace) {
    /* free storage for imap and recreate fxyz */
    /* disable everything, extra safe, and
encourage memory compaction *

call dodisable(0);
call dogsync();
call dodestroy( IGimap );
call dogsync();

ctype = 'real' // char(0);
gsize = 3*natoms; psize = 3*pagesize; blocksize = 1;
name = 'fxyz(3,natoms)' // char(0);
call dodeclare(IGfxyz, name, gsize, ctype, psize, blocksize);
call dogsync();
};

/* use a "pull" strategy to copy the data */

#ifdef USE_PROFILE
call profstart('copyxyz:cp xyz');
#endif

/* should be local, no need to search cache */
call dodisable( IGnumlist );
call doenable( IGxyz );
call dodisable( IGfxyz );

/* permuted copy xyz to fxyz, then fxyz to xyzlist */

gsize = natoms;
npages = int( gsize/pagesize ) + 1;
ipagestrt=myid;
doloop4(ipage,ipagestrt,npages,nproc) {
  istrt = 1 + ipage*pagesize;
iend = min(gsize, istrt+pagesize-1);
isize = iend - istrt + 1;
  if (isize <= 0) break;
  call dooait( dobigather( IGnumlist, isize, idxnumlist(istrt), numlist ));

doloop(i,1,isize) {
  ni = numlist(i);
  jp = idxyz(i,ni);
  /* assume continuous storage */
  iposxyz(1,i) = jp;
iposxyz(2,i) = jp+1;
iposxyz(3,i) = jp+2;
};
call dorgather( IGxyz, 3*isize, iposxyz, xyz );
call dobrcatter( IGfxyz, 3*isize, idxfxyz(1,istrt), xyz );
* end do ipage */

#ifdef USE_PROFILE
    call profend('copyxyz:cp xyz');
#endif

    call dogsync();
    /* safer to copy vectors than renaming them, will simply swap
       global array descriptors to rename them if this is really costly. */
    call dodisable(IGxyzlist);
    gsize = 3*natoms; psize = 3*pagesize;
    call gvveccopy(IGfxyz, gsize, psize, IGxyzlist);
    call dogsync();
    /* permuted copy vxyz to fxyz, then fxyz to vxyz */
    call doenable(IGvxyz);
    call dodisable(IGfxyz);

#ifdef USE_PROFILE
    call profstart('copyxyz:cp vel');
#endif

    gsize = natoms;
    npages = int( gsize / pagesize ) + 1;
    ipagestrt=myid;

doloop4(ipage,ipagestrt,npages,nproc) {
    istrt = 1 + ipage*pagesize;
    iend = min(gsize, istrt+pagesize-1);
    isize = iend - istrt + 1;
    if (isize <= 0) { break; };
    call dowait( dobigather(IGnumlist, isize, idxnumlist(istrt), numlist ));

doloop(i,l,isize) {
    ni = numlist(l);
    jp = idxxyz(1,ni);
    /* assume continuous storage */
    iposxyz(1,i) = jp;
    iposxyz(2,i) = jp+1;
    iposxyz(3,i) = jp+2;
    }
    call dorgatherc(IGvxyz, 3*isize, iposxyz, vxyz);
    call dobscatter(IGfxyz, 3*isize, idxfxyz(l,istrt), vxyz );

    } /* end do ipage */

#ifdef USE_PROFILE
    call profend('copyxyz:cp vel');
#endif

    call dogsync();
 safer to copy vectors than renaming them, will simply swap
global array descriptors to rename them if this is really costly.
 */

call dodisable( IGvxyz );
gsize = 3*natoms; psize = 3*pansize;
call grvecopy( IGfxyz, gsize, psize, IGvxyz );
call dogsync();

if (useminsace & usedfxyz) {
    /* disable everything */
    call dodisable( 0 );
call dogsync();
call dodeestroy( IGfxyz );

    /* recreate real*8 global fxyz array */
    ctype = 'real*8' // char(0);
gsize = 3*natoms; psize = 3*pansize; blocksize = 1;
    name = 'fxyz(3,natoms)' // char(0);
call dodeclare( IGfxyz, name, gsize, ctype, psize, blocksize );
call dogsync();
}

#ifdef USE_PROFILE
call profend('copyxyz')
#endif
return;
} end

4.8. subroutine force

#include "stdinc.h"

subroutine force()
#include "globals.h"
#include "parallel.h"
{
#define maxproc (2*1024)
    integer mystart(maxproc+1);
integer istrt, iend, ncols, binsize;
    logical islast;

integer gsize, psize;
real one, zero;
    parameter(one=1.0, zero=0.0);

real oned, zeroed;
    parameter(oned=1.0d0, zeroed=0.0d0);

real rvalue;
real dvalue;

/* perform dynamic load balancing and all force computations */

/* ============== start execution ============== */

#ifdef USE_PROFILE
    call profstart('force')
#endif

    ASSERT((1 <= nproc) & (nproc <= maxproc),
        ' ** force: invalid nproc ', nproc);

#ifdef NO_BALANCE
    /* no need to call "calwork" */
    call lbalance( mystart );
#else
    call profstart('force:calwork');
    call calwork();
    call lbalance( mystart );
    call profend('force:calwork');
#endif

    istrt = mystart(myid+1);

    /* iend is the last column actually computed */
    islast = ((myid + 1) == nproc);
    if (islast) {
        binsize = nbiny*nbinz;
        iend = binsize;
    } else {
        iend = mystart((myid+1) + 1) - 1;
    }

    ncols = (iend - istrt + 1);

    /* clear out array for accumulation */

    gsize = 3*natoms; psize = 3*pagesize;
#ifdef USE_DFXYZ
    dvalue = 0.0;
    call gdvecfill( IGfxyz, dvalue, gsize, psize );
#else
    rvalue = 0.0;
    call grvecfill( IGfxyz, rvalue, gsize, psize );
#endif
v = 0.0; w = 0.0;

if (ncols > 0) {
    call forceall( istrt, ncols);
}

call dogsync();

/* incorporate energy factors */

v = 4.0*v;

w = 48.0*w/3.0;

gsize = 3*natoms; psize = 3*pagesize;

#include "stdinc.h"

#ifdef USE_SYMMETRY
    nneibor = 14
#endif

#else
    nneibor = 27
#endif

}while

if_defined USEPROFILE
    call profend( 'force' )
#endif

return;

end

4.9. subroutine forceall

#include "stdinc.h"

#ifdef USE_SYMMETRY

define nneibor 14
#endif

#else

define nneibor 27
#endif

#ifdef USE_SYMMETRY

define ncols 5
#endif

#else

define ncols 9
#endif
subroutine forceall(ipstart, ncolumns)
#include "globals.h"
#include "parallel.h"

integer ipstart, ncolumns;
{
real one, zero; parameter(one=1.0, zero=0.0);

integer jj, ipos, ix, iy, iz, ijob;
integer icol, i, ntotal;
integer gsize, psize;
logical isnewcol, isfirst;
integer iyold;

/* =========
          start execution ========= */
#endif

call profstart('forceall')
#endif

call dodisable(IGfxyz);
call doenable(IGxyzlist);
call doenable(IGncount);
call doenable(IGxlist);

iyold = -1; isfirst = TRUE;
doloop(jj, 1, ncolumns) {
  ipos = ipstart + (jj-1);
  /* convert ipos back to (iy, iz) */
  call ni2ij(ipos, nbiny, nbinz, iy, iz);

  isnewcol = (iyold + 1 != iy);
  if (isfirst) { ijob = 1; isfirst = FALSE; }
  else if (isnewcol) { ijob = 2; } else { ijob = 3; }
  call forcecol(iy, iz, ijob);

  iyold = iy;
}

/* final flushing of results */

ijob = 4;
call forcecol(iy, iz, ijob);
4.10. subroutine forcecol

#include "stdinc.h"

#undef USEPROFILE
#define logdev 16
#define eps 1.0e-5
#define isapprox(r1,r2) ((abs((r1)-(r2)) <= eps*max(one,max(abs(r1),abs(r2)))))
#define RETURN_LABEL 999

#ifdef USE_SYMMETRY
#endif

#define CHECKXYZ(icol) { \
   doloop(jx,1,nbx) { \
      \n      call doigather( IGncount, 1,idxncount(jx,jy,jz),icount); \
      ASSERT( icount == ncount(jx,icol), \
         ** forcecol: icount != ncount(jx,icol), icount); \
      } 
   } 

#ifdef USE_SYMMETRY
#endif
\[\]
\[
\begin{align*}
\text{jp} &= \text{xlist}(\text{icol}); \\
\text{doloop}(j,1,j+1) \{ \text{jp} = \text{jp} + \text{ncount}(j,\text{icol}); \}; \\
\text{call doigather( IGxlist, 1, idxxlist}(jy, jy, jz), \text{ip}); \\
\text{ASSERT( ip == } \text{jp, \}} \\
\quad \text{** forcecol: error in xlist calculation }', \text{jp}); \\
\text{if (icount > 0) } \{ \text{call dowait( doigrather( IGxlist, 3*icount, \\
\quad \text{idxxlist}(1,ip), txyz )); \}
\end{align*}
\]

\[
\begin{align*}
\text{doloop}(j,1,icount) \{ \text{jp} &= \text{gxlist}(jx,\text{icol}) + (j-1); \\
\text{ASSERT( isapprox(tx(j),rx(jp)), \}
\quad \text{** forcecol: tx(j) != rx(jp), icol=',icol); \}
\text{if (icount > 0) \}
\end{align*}
\]

\[
\begin{align*}
\text{isrc} &= \text{gxlist}(0,\text{icol}); \text{idest} = \text{gxlist}(\text{nbinx},\text{icol}); \text{icount} = \text{ncount}(0,\text{icol}); \\
\text{doloop}(1,1,icount) \{ \text{fx}(\text{idest}) = \text{fx}(\text{idest}) + \text{fx}(\text{isrc}); \\
\quad \text{fy}(\text{idest}) = \text{fy}(\text{idest}) + \text{fy}(\text{isrc}); \\
\quad \text{fz}(\text{idest}) = \text{fz}(\text{idest}) + \text{fz}(\text{isrc}); \\
\text{idest} &= \text{idest} + 1; \text{isrc} = \text{isrc} + 1; \\
\}
\end{align*}
\]

\[
\begin{align*}
\text{isrc} &= \text{gxlist} (\text{nbinx}+1,\text{icol}); \text{idest} = \text{gxlist}(1,\text{icol}); \text{icount} = \text{ncount}(\text{nbinx}+1,\text{icol}); \\
\text{doloop}(1,1,icount) \{ \text{fx}(\text{idest}) = \text{fx}(\text{idest}) + \text{fx}(\text{isrc}); \\
\quad \text{fy}(\text{idest}) = \text{fy}(\text{idest}) + \text{fy}(\text{isrc}); \\
\quad \text{fz}(\text{idest}) = \text{fz}(\text{idest}) + \text{fz}(\text{isrc}); \\
\text{idest} &= \text{idest} + 1; \text{isrc} = \text{isrc} + 1; \\
\}
\end{align*}
\]

\[
\begin{align*}
\text{#define FOLDBC( icol ) } \{ \text{ /* fold boundary contribution back */ } \\
\text{isrc} &= \text{gxlist}(0,\text{icol}); \text{idest} = \text{gxlist}(\text{nbinx},\text{icol}); \text{icount} = \text{ncount}(0,\text{icol}); \\
\text{doloop}(1,1,icount) \{ \text{fx}(\text{idest}) = \text{fx}(\text{idest}) + \text{fx}(\text{isrc}); \\
\quad \text{fy}(\text{idest}) = \text{fy}(\text{idest}) + \text{fy}(\text{isrc}); \\
\quad \text{fz}(\text{idest}) = \text{fz}(\text{idest}) + \text{fz}(\text{isrc}); \\
\text{idest} &= \text{idest} + 1; \text{isrc} = \text{isrc} + 1; \\
\}
\end{align*}
\]

\[
\begin{align*}
\text{#define FLUSHOUT( icol ) } \{ \text{ /* end do j */ } \\
\text{if (ntotal(ico1) > 0) } \{ \text{FOLDBC( icol ); \\
\text{if (usedfxyz) } \}
\end{align*}
\]


call dobraxpy(IGfxyz, 3 * ntotal(icol), \n    idfxyz(1, xlist(icol)), \n    fxyz(1, gxlist(1, icol)), one, one); 
}; 

#define SHIFTCOPY( isrc, idest ) { 
    isvalid = (1 <= isrc) & (isrc <= ncols); 
    ASSERT( isvalid, \n        ' ** forcecol: invalid isrc ', isrc ); 
    isvalid = (1 <= idest) & (idest <= ncols); 
    ASSERT( isvalid, \n        ' ** forcecol: invalid idest ', idest ); 
    nlist(1, idest) = nlist(1, isrc); 
    nlist(2, idest) = nlist(2, isrc); 
    ntotal(idest) = ntotal(isrc); 
    xlist(idest) = xlist(isrc); 
    ipdest = gxlist(0,idest); 
    ASSERT( ipdest == ipstart(idest), \n        ' ** forcecol: ipdest != ipstart(idest) ',idest ); 
    ipsrc = gxlist(0,isrc); 
    ASSERT( ipsrc == ipstart(isrc), \n        ' ** forcecol: ipsrc != ipstart(isrc) ',isrc ); 
    gntotal = ntotal(idest) + ncount(0,idest) + ncount(nbinx+1,idest); 
    call icopy(nbinx+2, gxlist(0, isrc), 1, gxlist(0, idest), 1); 
    call icopy(nbinx+2, ncount(0, isrc), 1, ncount(0, idest), 1); 
    /* reset ipstart(:) to reuse storage in xyz(:) and fxyz(:) */ 
    ipstart(idest) = ipsrc; 
    ipstart(isrc) = ipdest; 
    gxlist(0,isrc) = ipstart(isrc); 
    gxlist(0,dest) = ipstart(idest); 
    if (IDEBUG >= 2) { 
        call rfill( 3*ndim, zero, xyz(1,ipstart(isrc)), 1 ); 
        if (usedfxyz) { 
            call dfill( 3*ndim, zero, fxyz(1,ipstart(isrc)), 1 ); 
        } else { 
            call rfill( 3*ndim, zero, fxyz(1,ipstart(isrc)), 1 ); 
        } 
    } else if (IDEBUG) */ 
};
compute forces associated with a column (ix=1..nbinx) for a particular (iy,iz) /*
#define gndim (ncols*ndim) subroutine forcecol(iy, iz, ijob)
#include "globals.h"
#include "parallel.h"
integer iy, iz;
integer ijob;
{
#ifdef MP
#define nthread 6 */
/* integer numcpus; external numcpus; */
/* #define nthread (numcpus()) */
#define nthread 2
#else
#define nthread 1
#endif
integer jzstart,ixend,istep;

logical usesymmetry;
#endif USE_SYMMETRY
 parameter(usesymmetry=TRUE);
#else
 parameter(usesymmetry=FALSE);
#endif
integer maxbinx, ndim;
#if R8||i860
/* R8 has less memory, unlikely to solve extremely huge problems */
 parameter(maxbinx=64, ndim=700);
#else
 parameter(maxbinx=700, ndim=20*maxbinx);
#endif
integer ncount(0:(maxbinx+1), ncols);
integer gxlist(0:(maxbinx+1), ncols);
integer xlist(ncols);

#endif USE_DFXYZ
#define REAL real8
#else
#define REAL real
#endif
REAL vvpos,vpos(maxbinx),vvneg,vneg(maxbinx);
REAL wupos,wpos(maxbinx),wneg,wneg(maxbinx);
real txyz(3,ndim);
#define tx(i) txyz(1,i)


#define ty(i) txyz(2,i)
#define tz(i) txyz(3,i)

logical usedfxyz;

#ifdef USE_DFXYZ
    parameter(usedfxyz=TRUE);
    real8 fxyz(3, gndim);
#else
    parameter(usedfxyz=FALSE);
    real fxyz(3, gndim);
#endif

real xyz(3, gndim);

#define rx(ip) xyz(1,ip)
#define ry(ip) xyz(2,ip)
#define rz(ip) xyz(3,ip)
#define fx(ip) fxyz(1,ip)
#define fy(ip) fxyz(2,ip)
#define fz(ip) fxyz(3,ip)

real8 oned,zerod;
parameter(oned=1.0d0,zerod=1.0d0);
real one, zero;
parameter(one=1.0,zero=0.0);

integer maxcols;
parameter(maxcols = 9);
integer list(maxcols), nlist(2, maxcols);

integer lcount(nneibor, maxbinx), lxlist(nneibor, maxbinx);
integer gntotal, ntotal(maxcols);
real xadj, yadj, zadj, yadjust(maxcols), zadjust(maxcols);

integer i, j, ip, jp, istrt, iend;
integer jcol, icol, icolstrt, icolend, icolsize;
integer ix, jx, jy, ja, id, idy, idz, jdx, jdy, jdz;
integer isrc, idest, icount;
logical alloc, isvalid, spaceok;
logical isfirst, isnewcol, isshift, islast;
integer ipstart(maxcols), ipsrc, ipdest;

save ipstart, ncount, xlist, glist, nlist, ntotal;
save xyz, vxyz, fxyz;

#ifdef USE_SYMMETRY
#define ncopies 3
#else
#define ncopies 6
#endif
    integer nclist(2,ncopies); save nclist;

#include "mapping.h"

data isfirst / TRUE /;
#endif USE_SYMmetry

/* copy sequence 4->1, 3->2, then 5->3 */
/* based on the ordering: */
[ *, *, 2]
[ *, 1, 3]
[ *, 4, 5]
/*
data nclist / 4,1, 3,2, 5,3 /;
#else
/* copy sequence 4->1, 5->2, 6->3, then 7->4, 8->5, 9->6 */
/* based on the ordering: */
[ 1, 2, 3]
[ 4, 5, 6]
[ 7, 8, 9]
/*
data nclist / 4,1, 5,2, 6,3, 7,4, 8,5, 9,6 /;
#endif USE_SYMmetry */

/* =========== start execution =========== */

dir$r noconcur

#endif USE_PROFILE
    call profstart('forcecol');
#endif

spaceok = (1 <= nbinx) & (nbinx <= maxbinx);
ASSERT(spaceok,
    ' ** forcecol: invalid nbinx ', nbinx);
isvalid = (1 <= ijob) & (ijob <= 4);
ASSERT( invalid,
    ' ** forcecol: invalid ijob ', ijob);
isfirst = (ijob == 1);
isnewcol = (ijob == 2);
isshift = (ijob == 3);
islast = (ijob == 4);
if (isfirst) {
    /* initialize data structure */
    doloop(icols,1,ncols) {
        ip = (icol-1)*ndim + 1;
        ipstart(icol) = ip;
    };
};

if (!isshift) {
    /* get everything from scratch */
    icolstrt = 1;
} else {
    /* reuse previous data */
    if (usesymmetry) { icolstrt = ncols - 1; } else { icolstrt = ncols - 2; };
};

icolend = ncols;
icolsize = (icolend - icolstrt + 1);

/* flush previous entries ? Nothing to flush out if it isfirst */
if (isshift) {
    if (usesymmetry) {
        icol = 1; FLUSHOUT(icol);
        icol = 2; FLUSHOUT(icol);
    } else {
        /* fxyz of neighbours are not updated, flush only myown */
        icol = 5; FLUSHOUT(icol);
    };
} else if (isnewcol | islast) {
    if (usesymmetry) {
        /* flush out everything */
        doloop(icols,1,ncols) {
            FLUSHOUT(icol);
        };
    } else {
        /* fxyz of neighbours are not updated, flush only myown */
        icol = 5; FLUSHOUT(icol);
    };
};

/* perform shifts */
if (isshift) {

```c

doloop(i,1,ncopies) {
    isrc = nclist(1,i);
    idest = nclist(2,i);
    SHIFTCOPY( isrc, idest );
};

/* bring in xyz coordinates, and clear out fxyz */
if (islast) { goto RETURN_LABEL; /* don't do any more work */ ;

doloop(icol, 1, ncols) {
    yadjust(icol) = 0.0;
    zadjust(icol) = 0.0;
};

/* determine neigbors */
doloop(icol, icolstrt, icolend) {
    idy = colmap(1, icol);
    idz = colmap(2, icol);
    jy = iy + idy;
    jz = iz + idz;
    if (jy > nbiny) { jy = jy - nbiny; yadjust(icol) = one; }
    if (jy < 1) { jy = jy + nbiny; yadjust(icol) = -one; }
    if (jz > nbinz) { jz = jz - nbinz; zadjust(icol) = one; }
    if (jz < 1) { jz = jz + nbinz; zadjust(icol) = -one; }
    nlist(1, icol) = jy;
    nlist(2, icol) = jz;
};

/* get xlist(:) */
jx = 1;
doloop(icol, icolstrt, icolend) {
    jy = nlist(1, icol);
    jx = nlist(2, icol);
    list(icol) = idxxlist(jx, jy, jz);
};
call doenable(IGxlist);
call doigather(IGxlist, icolsize, list(icolstrt), xlist(icolstrt));

/* get ncount(:) for columns */
call doenable( IGNcount );
doloop(icol, icolstrt, icolend) {
```
jx = 1;
jjy = nlist(1, icol);
jjz = nlist(2, icol);
call dowait(dobigather(kGncount, nbincx,
idxncount(jx, jjy, jjz), ncount(1, icol)));

if (DEBUG >= 2) {
    /* double check */
    for (ix=1; ix<=nbincx; ix++)
        call doigather( kGncount, 1, idxncount(ix,jy,jz), icount);
    ASSERT( icount != ncount(icol),
            /* forcecol: icount != ncount(icol)',ix */);
};

/* wrap around boundary condition */
ncount(0, icol) = ncount(nbincx, icol);
ncount(nbincx + 1, icol) = ncount(1, icol);

/* calculate total storage needed */
gntotal = 0;
do(i, 1, nbincx) {
    gntotal = gntotal + ncount(i, icol);
};
ntotal(icol) = gntotal;

ntotal(icol) = ntotal = ncount(0, icol) + ncount(nbincx + 1, icol);
spaceok = (0 <= gntotal) & (gntotal <= ndim);
ASSERT(spaceok,
        /* forcecol: insufficient temporary space, need ', gntotal); */);

/* set up gxlist(:) */
do(icol, icolstrt, icolend) {
    gxlist(0,icol) = ipstart(icol);
    /* not tightly packed, make room for shifting columns */
do(i, 0, nbincx) {
        gxlist(i + 1, icol) = gxlist(i, icol) + ncount(i, icol);
    };
};

/* double check ipstart(:) and gxlist(:) */
if (DEBUG >= 2) {
    do(icol,1,ncols) {
        ASSERT( ipstart(icol) == gxlist(0,icol),
                /* forcecol: invalid ipstart(icol)',icol ); */
    };
    do(icol,1,ncols) {
        do(jcol,icol+1,ncols) {
            ASSERT( ipstart(icol) != ipstart(jcol),
                    /* forcecol: duplicate entry in ipstart():',icol ); */
        };
    };
};

/* get xyz coordinates */
do(icol, icolstrt, icolend) {
jy = nlist(1, icol);
jz = nlist(2, icol);

if (ntotal(icol) > 0) {
call dowait(dobrgather(1Gxyzlist, 3 * ntotal(icol),
   idxyzlist(1, xlist(icol)), xyz(1, gxlist(1, icol))));

   if (IDEBUG >= 2) {
      /* double check xyz coordinates */
      CHECKXYZ( icol );
   } /* if (DEBUG) */
};

/* fix up boundary */
isrc = gxlist(nbinx, icol);
idest = gxlist(0, icol);
icount = ncount(0, icol);

istrt = idest;
ien = istrt + (icount - 1);
doloop(idest, istrt, ien) {
   rx(idest) = rx(isrc) - one;
   ry(idest) = ry(isrc);
   rz(idest) = rz(isrc);

   src = src + 1;
};

isrc = gxlist(1, icol);
idest = gxlist(nbinx + 1, icol);
icount = ncount(nbinx + 1, icol);

istrt = idest;
ien = istrt + (icount - 1);
doloop(idest, istrt, ien) {
   rx(idest) = rx(isrc) + one;
   ry(idest) = ry(isrc);
   rz(idest) = rz(isrc);

   src = src + 1;
};

/* adjust y,z coordinates */
doloop(icol, icolstr, icolend) {
   gtotal = ntotal(icol) + ncount(0, icol) + ncount(nbinx + 1, icol);
   istrt = gxlist(0, icol);
   iend = istrt + gtotal - 1;

   if (yadjust(icol) != zero) {
      yadj = yadjust(icol);
      doloop(1, istrt, iend) {
         ry(i) = ry(i) + yadj;
      };
   }
if (zadjust(icol) != zero) {
    zadj = zadjust(icol);
    doloop(i, istrt, iend) {
        rz(i) = rz(i) + zadj;
    }
};
}

/* zero out fxyz */
if (usesymmetry) {
    doloop(icol,icolstrt,icolend) {
        gntotal = ntotal(icol) + ncount(0, icol) + ncount(nbinx + 1, icol);
        istrt = gxlist(0, icol);
        iend = istrt + gntotal - 1;
        doloop(i,istrt,iend) {
            fx(i) = 0.0;
            fy(i) = 0.0;
            fz(i) = 0.0;
        }
    };
} /* end do icol */
else {
    /* zero out only 1 column of fxyz */
    icol = 5;
    gntotal = ntotal(icol) + ncount(0, icol) + ncount(nbinx + 1, icol);
    istrt = gxlist(0, icol);
    iend = istrt + gntotal - 1;
    doloop(i,istrt,iend) {
        fx(i) = 0.0;
        fy(i) = 0.0;
        fz(i) = 0.0;
    }
}
};

/* prepare lcount and lxlist */
doloop(ix,1,nbinx) {
    doloop(i, 1, nnneibor) {
        idx = mapping(1, i);
        idy = mapping(2, i);
        idz = mapping(3, i);
        jx = ix + idx;
        jy = iy + idy;
        jz = iz + idx;
        icol = icolmap(i);
        isvalid = (0 <= jx) & (jx <= nbinx + 1);
if (DEBUG >= 2) {
    ASSERT(isvalid, 
        '** forcecol: invalid jx ', jx):
    ASSERT((1 <= icol) & (icol <= ncols), 
        '** forcecol: invalid icol ', icol);
} /* if (DEBUG) */

jdy = colmap(1, icol);
jdz = colmap(2, icol);
if (DEBUG >= 2) {
    ASSERT((jdy == idy) & (jdz == idz),
        '** forcecol: invalid icol ', icol);
} /* if (DEBUG) */

lcount(i, ix) = ncount(jx, icol);
lxlist(i, ix) = gxlist(jx, icol);
}; /* end do i */

}; /* end do ix */

/* perform computation in routine 'forcecal' */

if (mod(nbinx,nthread) == 0) {
    /* evenly divide */
    istep = nbinx/nthread;
} else {
    istep = 1 + (nbinx/nthread);
};
ixend = istep;
doloop(ixstart,1,ixend) {
    call docheck();
#dir$l concur
#dir$l cncall
doloop4(ix,ixstart,nbinx,istep) {
    call forcecal(ix, iy, iz,
        lcount(i,ix), lxlist(i,ix),
        gndim, fxyz, xyz,
        vpos(i), vneg(i), vpos(i), vneg(i));
} /* end do ix */
}; /* end do ixstart */

vvpos = 0.0; vneg = 0.0; wspo = 0.0; wneg = 0.0;
doloop(ix,1,nbinx) {
    vvpos = vvpos + vpos(ix);
    vneg = vneg + vneg(ix);
    wspo = wspo + vpos(ix);
    wneg = wneg + vneg(ix);
};

if (w > 0.0) { w = (w + wspo) + wneg; } else { w = (w + wneg) + vpos; }
if (v > 0.0) { v = (v + vpos) + vneg; } else { v = (v + vneg) + vpos; }
RETURN_LABEL {
    ifdef USE_PROFILE
        call profend('forcecol');
    endif

    return;
};
}

4.11. subroutine forcecal

#include "stdinc.h"
#undef ORIGINAL
#define USE_ABS
#ifdef HP
#undef USE_PROFILE
#endif

#define RETURN_LABEL 9999
#ifdef USE_SYMMETRY
#ifdef nneibor
#define nneibor 14
#endif
#else
#ifdef nneibor
#define nneibor 27
#endif
#endif

#define maxnneibor 27

subroutine forcecal(ix,iy,iz,
    ncount, lxlist,
    ndim, fxyz, xyz,
    vpos, vneg, wpos, wneg)

#include "globals.h"

    integer ix,iy,iz;
    integer ncount(nneibor), lxlist(nneibor);
    integer ndim;
    real xyz(3, ndim);
ifdef USE_DFXYZ
    real8 fxyz(3,ndim);
define REAL real8
else
    real fxyz(3,ndim);
define REAL real
endif

{ define cdim 1024
    integer ii,ccount, ilist(cdim), jlist(cdim);
    REAL rsq(cdim);

    intrinsic sqrt, abs, max, min, mod, sign;
    integer ni, i, j, ip, jp, jstrt, jend, idxval;
    logical isempty;
    integer ip1, ipj, jneibor, jcount;
    integer ninside;
    logical isxok, isyok, iszok, isallok;
    integer idx, idy, idz, jx, jy, jz;
    REAL one, zero;
    parameter(one=1.0, zero=0.0);
    REAL rx, ry, rzi, rxj, ryj, rzj;
    REAL rxij, ryij, rzij, rij, rijsq;
    REAL s2, s6;
    REAL sigsq, rcutsq;
    REAL fij, fxij, fyij, fzij;
    REAL fxi, fyi, fzi;
    REAL wpos, wneg, vij, vpos, vneg;

    /* routine to compute forces and potential related to
       a single bin cell (ix, iy, iz) */

    #include "dolib.h"

    #define rx(i) xyz(1,i)
    #define ry(i) xyz(2,i)
    #define rz(i) xyz(3,i)
    #define fx(i) fxyz(1,i)
    #define fy(i) fxyz(2,i)
    #define fz(i) fxyz(3,i)

    /* ================ */

    #dir$r noconcur
ifdef USEPROFILE
exe$ CALL MP_BCS
call profstart('forcecal')
exe$ CALL MP_BCS
endif

isxok = (1 <= ix) & (ix <= nbinx);
isyok = (1 <= iy) & (iy <= nbiny);
iszok = (1 <= iz) & (iz <= nbinz);
isallok = isxok & isyok & iszok;
if (!isallok) {
    write(*,*), ** forcecal: invalid (ix,iy,iz) ',
        ix,iy,iz;
    stop ' ** ERROR in forcecal ';
};

ninside = ncount(1);
isempty = (ninside == 0);
if (isempty) { goto RETURN_LABEL; }

/* Assume fx(:,),fy(:,),fx(:,), already initialized */

vpos = zero; vneg = zero;

sigsq = sigma*sigma;
rcutsq = rcut*rcut;

/* note minus one for offset calculations */
ip1 = llist(1)-1;

/* handle interactions within the same cell */
ASSERT( cdim >= ninside*ninside,
    ' ** forcecal: cdim too small, need ', ninside*ninside ');
ip = 1;
doloop(i,1,ninside) {
    doloop(j,i+l,ninside) {
        rsq(ip) = (rx(ip1+i) - rx(ip1+j))**2 +
            (ry(ip1+i) - ry(ip1+j))**2 +
            (rz(ip1+i) - rz(ip1+j))**2;
        ip = ip + 1;
    }
}; /* end do i */

ccount = 0; ip = 1;
doloop(i,1,ninside) {
    doloop(j,i+1,ninside) {
        rijsq = rsq(ip);
        if (rijsq <= rcutsq) {
            ccount = ccount + 1;
        }
ilist(ccount) = i; jlist(ccount) = j;
rsq(ccount) = rijsq;
}
ip = ip + 1;
}; /* end do i */

/* perform force and potential calculations */

doloop(ii, 1, ccount) {
i = ilist(ii); j = jlist(ii); rijsq = rsq(ii);

rij = sqrt(rijsq);

rxij = rx(ipi+i) - rx(ipi+j);
ryij = ry(ipi+i) - ry(ipi+j);
rzij = rz(ipi+i) - rz(ipi+j);

sr2 = sigsq/rijsq;
sr6 = sr2*sr2*sr2;

/* v is the total potential energy */
/* w is the total virial energy */
vij = sr6*(sr6-1.0)-vrcut-dvrc12*(rij-rcut);
wij = sr6*(sr6-0.5)+dvrcut*rij;
fij = wij/rijsq;

fxij = fij*rxij;
fyij = fij*ryij;
fzij = fij*rzij;

fx(ipi+i) = fx(ipi+i) + fxij;
fy(ipi+i) = fy(ipi+i) + fyij;
fz(ipi+i) = fz(ipi+i) + fzij;

fx(ipi+j) = fx(ipi+j) - fxij;
fy(ipi+j) = fy(ipi+j) - fyij;
fz(ipi+j) = fz(ipi+j) - fzij;

#ifdef USE_ABS
vpos = vpos + (abs(vij) + vij);
vneg = vneg + (vij-abs(vij));

wpos = wpos + (abs(wij)+wij);
wneg = wneg + (wij-abs(wij));
#else
if (vij > 0) { vpos = vpos + vij;} else {vneg = vneg + vij;};
if (wij > 0) { wpos = wpos + wij;} else {wneg = wneg + wij;};
#endif /* USE_ABS */

}; /* end do ii */

/* handle interactions with neighbor cells */
doloop(jneibor,2,neibor) {

    ipj = lxlist(jneibor)-1;
    jcount = ncount(jneibor);
    ASSERT( cdim >= jcount*ninside,
              ' ** forcecal: cdim too small, need ', jcount*ninside);

    ip = 1;
    dooloop(j,1,jcount) {
        dooloop(i,1,ninside) {
            rsq(ip) = (rx(ip1+i) - rz(ipj+j))**2 +
                      (ry(ip1+i) - ry(ipj+j))**2 +
                      (rz(ip1+i) - rz(ipj+j))**2;
            ip = ip + 1;
        }
    }
    icount = 0; ip = 1;
    dooloop(j,1,jcount) {
        dooloop(i,1,ninside) {
            rij = rsq(ip);
            if (rij <= rcutsq) {
                icount = icount + 1;
                ilist(icount) = i; jlist(icount) = j;
                rsq(icount) = rij;
            }
            ip = ip + 1;
        }
    }

    dooloop(ii,1,icount) {
        i = ilist(ii); j = jlist(ii); rij = rsq(ii);
        rij = sqrt(rij);
        rxij = rx(ip1+i) - rz(ipj+j);
        ryij = ry(ip1+i) - ry(ipj+j);
        rzij = rz(ip1+i) - rz(ipj+j);

        sr2 = sigsq/rij;
        sr6 = sr2*sr2*sr2;
        wij = (sr6*(sr6-1.0)-vrcut-dwrcut2*(rij-rcut))
             + (sr6*(sr6-0.5)+drcut+rij);
        fij = wij/rij;

        fxij = rij*rxij;
        fyij = rij*ryij;
        fzij = rij*rzij;

        fx(ip1+i) = fx(ip1+i) + fxij;
        fy(ip1+i) = fy(ip1+i) + fyij;
        fz(ip1+i) = fz(ip1+i) + fzij;
    }
}

#ifdef USE_SYMMETRY

    fx(ipj+j) = fx(ipj+j) - fxij;
    fy(ipj+j) = fy(ipj+j) - fyij;
#endif
\[ f_z(i,p+j) = f_z(i,p+j) - f_z(i,j); \]

```c
else /* USE_SYMTERM */
    v_{ij} = 0.5 \times v_{ij};
    w_{ij} = 0.5 \times w_{ij};
endif /* USE_SYMTERM */
```

```c
#define USE_ABS
vpos = vpos + (\text{abs}(v_{ij}) + v_{ij});
vneg = vneg + (v_{ij} - \text{abs}(v_{ij}));

wpos = wpos + (\text{abs}(w_{ij}) + w_{ij});
wneg = wneg + (w_{ij} - \text{abs}(w_{ij}));

#undef USE_ABS
```

```c
if (v_{ij} > 0.0) {vpos = vpos + v_{ij};} else {vneg = vneg + v_{ij};}
if (w_{ij} > 0.0) {wpos = wpos + w_{ij};} else {wneg = wneg + w_{ij};}
```

```c
#define USE_ABS
vpos = 0.5 \times vpos; vneg = 0.5 \times vneg;
wpos = 0.5 \times wpos; wneg = 0.5 \times wneg;
#undef USE_ABS
```

```c
/* summation into global v and w performed in forcecol */
```

```c
RETURN_LABEL { 
#define USE_PROFILE
#define CALL_MP_BCS
    call profend('forcecal')
#define CALL_MP_ECS
#endif
```

```c
return;
};
}
```

```c
end
```
5. References


INTERNAL DISTRIBUTION

1. B. R. Appleton
2. B. A. Carreras
3-7. E. F. D’Azevedo
8. T. S. Darland
9. J. J. Dongarra
10. J. B. Drake
11. T. H. Dunigan
12. W. R. Emanuel
13. G. A. Geist
14. K. L. Kliever
15-19. M. R. Leuze
20. E. G. Ng
21. C. E. Oliver
22. B. W. Peyton
23-27. S. A. Raby
28. B. A. Riley
29-33. C. H. Romine
34. W. A. Shelton
35-39. R. F. Sincovec
40. G. M. Stocks
41. M. R. Strayer
42. D. W. Walker
43. P. H. Worley
44. T. Zacharia
45. Central Research Library
46. ORNL Patent Office
47. K-25 Applied Technology Library
48. Y-12 Technical Library
49. Laboratory Records - RC
50-51. Laboratory Records Dept.

EXTERNAL DISTRIBUTION

52. Loyce M. Adams, Applied Mathematics, FS-20, University of Washington, Seattle, WA 98195
53. Christopher R. Anderson, Department of Mathematics, University of California, Los Angeles, CA 90024
54. Todd Arbogast, Department of Mathematical Sciences, Rice University, P. O. Box 1892, Houston, TX 77251
55. Donald M. Austin, 6196 EECS Building, University of Minnesota, 200 Union Street, S.E., Minneapolis, MN 55455
56. Robert G. Babb, Oregon Graduate Center, CSE Department, 19600 N.W. Walker Road, Beaverton, OR 97006
57. David H. Bailey, NASA Ames, Mail Stop 258-5, NASA Ames Research Center, Moffet Field, CA 94035
58. Jesse L. Barlow, Department of Computer Science and Engineering, 220 Pond Laboratory, The Pennsylvania State University, University Park, PA 16802-6106
59. Edward H. Barsis, Computer Science and Mathematics, P. O. Box 5800, Sandia National Laboratory, Albuquerque, NM 87185
60. Adam Beguelin, Carnegie Mellon University, School of Computer Science, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890
61. Robert E. Benner, Parallel Processing Division 1413, Sandia National Laboratories, P. O. Box 5800, Albuquerque, NM 87185
62. Marsha J. Berger, Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012
63. Philippe Berger, Institut National Polytechnique, ENSEEIHT, 2 rue Charles Camichel-F, 31071 Toulouse Cedex, France
64. Ake Bjorck, Department of Mathematics, Linkoping University, S-581 83 Linkoping, Sweden
65. John H. Bolstad, L-16, Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94550
66. Roger W. Brockett, Harvard University, Pierce Hall, 29 Oxford Street Cambridge, MA 02138
67. James C. Browne, Department of Computer Sciences, University of Texas, Austin, TX 78712
68. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307
69. Donald A. Calahan, Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI 48109
70. Thomas A. Callcott, Director, The Science Alliance Program, 53 Turner House, University of Tennessee, Knoxville, TN 37996
71. Tony Chan, Department of Mathematics, University of California, Los Angeles, 405 Hilgard Avenue, Los Angeles, CA 90024
72. Jagdish Chandra, Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709
73. Siddhartha Chatterjee, Dept. of Computer Science, CB 3175, Sitterson Hall, The University of North Carolina, Chapel Hill, NC 27599-3175
74. Melvyn Ciment, National Science Foundation, 1800 G Street N.W., Washington, DC 20550
75. Tom Coleman, Department of Computer Science, Cornell University, Ithaca, NY 14853
76. Alva Couch, Department of Computer Science, Tufts University, Medford, MA 02155
77. Paul Concus, Mathematics and Computing, Lawrence Berkeley Laboratory, Berkeley, CA 94720
78. Tom Crockett, ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665-5225
79. Jane K. Cullum, IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598
80. George Cybenko, Center for Supercomputing Research and Development, University of Illinois, 104 South Wright Street, Urbana, IL 61801-2932
81. Helen Davis, Computer Science Department, Stanford University, Stanford, CA 94305
82. Michel Dayde, Institut National Polytechnique, ENSEEIHT, 2 rue Charles Camichel-F, 31071 Toulouse Cedex, France
83. Craig Douglas, IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598-0218
84. Donald J. Dudziak, Department of Nuclear Engineering, 110B Burlington Engineering Labs, North Carolina State University, Raleigh, NC 27695-7909
85. Iain S. Duff, Atlas Centre, Rutherford Appleton Laboratory, Chilton, Oxon OX11 0QX, England
86. Victor Eijkhout, University of Tennessee, 107 Ayres Hall, Department of Computer Science, Knoxville, TN 37996-1301
87. Stanley Eisenstat, Department of Computer Science, Yale University, P. O. Box 2158 Yale Station, New Haven, CT 06520
88. Howard C. Elman, Computer Science Department, University of Maryland, College Park, MD 20742
89. Richard E. Ewing, Director, Institute for Scientific Computations, Texas A&M University, College Station, TX 77843-3404
90. Edward Felten, Department of Computer Science, University of Washington, Seattle, WA 98195
91. Charles Fineman, Ames Research Center, Mail Stop 269/3, Moffet Field, CA 94035
92. David Fisher, Department of Mathematics, Harvey Mudd College, Claremont, CA 91711
93. Jon Flower, Parasoft Corporation, 2500 E. Foothill Boulevard, Suite 205, Pasadena, CA 91107
94. Geoffrey C. Fox, NPAC, 111 College Place, Syracuse University, Syracuse, NY 13244-4100
95. Chris Fraley, Statistical Sciences, Inc., 1700 Westlake Avenue N, Suite 500, Seattle, WA 98119
96. Joan M. Francioni, Computer Science Department, University of Southwestern Louisiana, Lafayette, LA 70504
97. Paul O. Frederickson, ACL, MS B287, Los Alamos National Laboratory, Los Alamos, NM 87545
98. Offir Frieder, George Mason University, Science and Technology Building, Computer Science Department, 4400 University Drive, Fairfax, VA 22030-4444
99. Robert E. Funderlic, Department of Computer Science, North Carolina State University, Raleigh, NC 27650
100. Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47401
101. C. William Gear, NEC Research Institute, 4 Independence Way, Princeton, NJ 08540
102. W. Morven Gentleman, Division of Electrical Engineering, National Research Council, Building M-50, Room 344, Montreal Road, Ottawa, Ontario, Canada K1A 0R8
103. Alan George, Vice President, Academic and Provost, Needles Hall, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
104. James Glimm, SUNY-Stony Brook, Department of Applied Mathematics and Statistics, Stony Brook, NY 11794
105. Gene Golub, Computer Science Department, Stanford University, Stanford, CA 94305
106. Joseph F. Grcar, Division 8245, Sandia National Laboratories, Livermore, CA 94551-0969
107. William D. Gropp, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
108. Eric Grosse, AT&T Bell Labs 2T-504, Murray Hill, NJ 07974
109. Sanjay Gupta, ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665-5225
110. John L. Gustafson, Ames Laboratory, 236 Wilhelm Hall, Iowa State University, Ames, IA 50011-3020
111. Christian Halloy, Assistant Director of JICS, 104 South College, Joint Institute for Computational Science, University of Tennessee, Knoxville, TN 37996-1301
112. Sven J. Hammarling, The Numerical Algorithms Group, Ltd., Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, United Kingdom
113. Robert M. Haralick, Department of Electrical Engineering, Director, Intelligent Systems Lab, University of Washington, 402 Electrical Engineering Building, FT-10, Seattle, WA 98195
114. Ann H. Hayes, Computing and Communications Division, Los Alamos National Laboratory, Los Alamos, NM 87545
115. Michael T. Heath, National Center for Supercomputing Applications, 4157 Beckman Institute University of Illinois, 405 North Mathews Avenue, Urbana, IL 61801-2300
116. Gerald W. Hedstrom, L-71, Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94550
117. Don E. Heller, Ames Laboratory, 327 Wilhelm, Ames, IA 50011
118. John L. Hennessy, CIS 208, Stanford University, Stanford, CA 94305
119. N. J. Higham, Department of Mathematics, University of Manchester, Gtr Manchester, M13 9PL, England
123. Robert E. Huddleston, Computation Department, Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94550
124. Jenq-Neng Hwang, Department of Electrical Engineering, FT-10, University of Washington, Seattle, WA 98195

125. Ilse Ipsen, Department of Computer Science, Yale University, P. O. Box 2158 Yale Station, New Haven, CT 06520

126. Leah H. Jamieson, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907


128. Lennart Johnsson, Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142-1214

129. Harry Jordan, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309

130. Bo Kagstrom, Institute of Information Processing, University of Umea, 5-901 87 Umea, Sweden

131. Malvyn Kalos, Cornell Theory Center, Engineering and Theory Center Building, Cornell University, Ithaca, NY 14853-3901

132. Hans Kaper, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Bldg. 221, Argonne, IL 60439

133. Alan H. Karp, HP Labs 3U-7, Hewlett-Packard Company, 1501 Page Mill Road, Palo Alto, CA 94304

134. Linda Kaufman, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

135. Robert J. Kee, Applied Mathematics Division 8245, Sandia National Laboratories, Livermore, CA 94551-0969

136. Kenneth Kennedy, Department of Computer Science, Rice University, P.O. Box 1892, Houston, TX 77001


138. Clyde P. Kruskal, Department of Computer Science, University of Maryland, College Park, MD 20742

139. Edward Kushner, Intel Corporation, 15201 NW Greenbrier Parkway, Beaverton, OR 97006

140. Michael Langston, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301

141. Richard Lau, Office of Naval Research, Code 111MA 800 Quincy Street, Boston Tower 1, Arlington, VA 22217-5000

142. Robert L. Launer, Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709

143. Tom Leighton, Lab for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139
144. James E. Leiss, Rt. 2, Box 142C, Broadway, VA 22815
145. Robert Leland, Sandia National Laboratories, 1424, P. O. Box 5800, Albuquerque, NM 87185-5800
146. Randall J. LeVeque, Applied Mathematics, FS-20, University of Washington, Seattle, WA 98195
147. John G. Lewis, Boeing Computer Services, P. O. Box 24346, M/S 7L-21, Seattle, WA 98124-0346
148. Heather M. Liddell, Center for Parallel Computing, Department of Computer Science and Statistics, Queen Mary College, University of London, Mile End Road, London E1 4NS, England
149. Brent Lindquist, SUNY-Stony Brook, Department of Applied Mathematics and Statistics, Stony Brook, NY 11794
150. Rik Littlefield, Pacific Northwest Laboratory, MS K1-87, P.O.Box 999, Richland, WA 99352
151. Joseph Liu, Department of Computer Science, York University, 4700 Keele Street, Downsview, Ontario, Canada M3J 1P3
152. Franklin Luk, Department of Computer Science, Amos Eaton Building — No. 131 Rensselaer Polytechnic Institute Troy, NY 12180-3590
153. Ewing Lusk, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, MCS 221 Argonne, IL 60439-4844
154. Allen D. Malony, Department of Computer and Information Science, University of Oregon, Eugene, OR 97403
155. Thomas A. Manteuffel, Department of Mathematics, University of Colorado - Denver, Denver, CO 80202
156. Anita Mayo, IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598
157. Oliver McBryan, University of Colorado at Boulder, Department of Computer Science, Campus Box 425, Boulder, CO 80309-0425
158. James McGraw, Lawrence Livermore National Laboratory, L-306, P. O. Box 808, Livermore, CA 94550
159. Piyush Mehrotra, ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665
160. Paul C. Messina, Mail Code 158-79, California Institute of Technology, 1201 E. California Boulevard, Pasadena, CA 91125
161. Cleve B. Moler, MathWorks, 325 Linfield Place, Menlo Park, CA 94025
162. Neville Moray, Department of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urbana, IL 61801
163. Jorge J. More, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
164. William A. Mulder, Koninklijke Shell Exploratie en Produktie Laboratorium, Postbus 60, 2280 AB Rijswijk, The Netherlands
166. V. E. Oberacker, Department of Physics, Vanderbilt University, Box 1807, Station B, Nashville, TN 37235
167. Dianne P. O’Leary, Computer Science Department, University of Maryland, College Park, MD 20742
168. Joseph Oliger, Computer Science Department, Stanford University, Stanford, CA 94305
169. James M. Ortega, Department of Computer Science, Thornton Hall, University of Virginia, Charlottesville, VA 22901
170. Steve Otto, Oregon Graduate Institute, Department of Computer Science and Engineering, 19600 NW von Neumann Drive, Beaverton, OR 97006-1999
171. Cherri Pancake, Department of Computer Science, Oregon State University, Corvallis, OR 97331-3202
172. Joseph E. Pasciak, Applied Mathematics, Brookhaven National Laboratory, Upton, NY 11973
173. Merrell Patrick, National Science Foundation, 1800 G Street N.W., Washington, DC 20550
174. David Payne, Intel Corporation, Supercomputer Systems Division, 15201 NW Greenbrier Parkway, Beaverton, OR 97006
175. Ronald F. Peierls, DAS — Bldg. 490-D, P.O. Box 5000, Brookhaven National Laboratory, Upton, NY 11973
176. Linda R. Petzold, Computer Science Department, University of Minnesota, 200 Union Street, S.E., Room 4-192, Minneapolis, MN 55455
177. Dan Pierce, Boeing Computer Services, P.O. Box 24346, M/S 7L-21, Seattle, WA 98124-0346
178. Paul Pierce, Intel Corporation, Supercomputer Systems Division, 15201 NW Greenbrier Parkway, Beaverton, OR 97006
179. Robert J. Plemmons, Departments of Mathematics and Computer Science, North Carolina State University, Raleigh, NC 27650
180. James C. T. Pool, Deputy Director, Caltech Concurrent Supercomputing Facility, California Institute of Technology, MS 158-79, Pasadena, CA 91125
181. Jesse Poore, Computer Science Department, University of Tennessee, Knoxville, TN 37996-1300
182. David A. Poplawski, Department of Computer Science, Michigan Technological University, Houghton, MI 49931
183. Roldan Pozo, University of Tennessee, 107 Ayres Hall, Department of Computer Science, Knoxville, TN 37996-1301
184. Padma Raghavan, University of Illinois, NCSA, 4151 Beckman Institute, 405 North Matthews Avenue, Urbana, IL 61801
185. Daniel A. Reed, Computer Science Department, University of Illinois, Urbana, IL 61801
186. John K. Reid, Numerical Analysis Group, Central Computing Department, Atlas Centre, Rutherford Appleton Laboratory, Didcot, Oxon OX11 0QX, England
187. John R. Rice, Computer Science Department, Purdue University, West Lafayette, IN 47907
188. Garry Rodrigue, Numerical Mathematics Group, Lawrence Livermore National Laboratory, Livermore, CA 94550
189. Donald J. Rose, Department of Computer Science, Duke University, Durham, NC 27706
190. Ahmed H. Sameh, Department of Computer Science, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455
191. Joel Saltz, Computer Science Department, A.V. Williams Building, University of Maryland, College Park, MD 20742
192. Jorge Sanz, IBM Almaden Research Center, Department K53/802, 650 Harry Road, San Jose, CA 95120
193. Robert B. Schnabel, Department of Computer Science, University of Colorado at Boulder, ECOT 7-7 Engineering Center, Campus Box 430, Boulder, CO 80309-0430
194. Robert Schreiber, RIACS, MS 230-5, NASA Ames Research Center, Moffett Field, CA 94035
195. Martin H. Schultz, Department of Computer Science, Yale University, P. O. Box 2158 Yale Station, New Haven, CT 06520
196. David S. Scott, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006
197. The Secretary, Department of Computer Science and Statistics, The University of Rhode Island, Kingston, RI 02881
198. Charles L. Seitz, Department of Computer Science, California Institute of Technology, Pasadena, CA 91125
199. Margaret L. Simmons, Computing and Communications Division, Los Alamos National Laboratory, Los Alamos, NM 87545
200. Horst D. Simon, NASA Ames Research Center, Mail Stop T045-1, Moffett Field, CA 94035
201. William C. Skamarock, 3973 Escuela Court, Boulder, CO 80301
202. Tony Skjellum, Dept of Computer Science, Mississippi State University, PO Drawer CS, Mississippi State, MS 39762-5623
203. Burton Smith, Tera Computer Company, 400 North 34th Street, Suite 300, Seattle, WA 98103
204. Marc Snir, IBM T.J. Watson Research Center, Department 420/36-241, P. O. Box 218, Yorktown Heights, NY 10598
205. Larry Snyder, Department of Computer Science and Engineering, FR-35, University of Washington, Seattle, WA 98195
206. Danny C. Sorensen, Department of Mathematical Sciences, Rice University, P. O. Box 1892, Houston, TX 77251

207. Rick Stevens, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

208. G. W. Stewart, Computer Science Department, University of Maryland, College Park, MD 20742

209. Paul N. Swarztrauber, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307

210. Julie Swisshelm, Sandia National Laboratories, 1421, Parallel Computational Sciences Department, Albuquerque, New Mexico 87185-5800

211. Wei Pai Tang, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

212. Bernard Tourancheau, LIP ENS-Lyon 69364, Lyon cedex 07, France

213. Joseph F. Traub, Department of Computer Science, Columbia University, New York, NY 10027

214. Lloyd N. Trefethen, Department of Computer Science, Cornell University, Ithaca, NY 14853

215. Robert van de Geijn, University of Texas, Department of Computer Sciences, TAI 2.124, Austin, TX 78712

216. Charles Van Loan, Department of Computer Science, Cornell University, Ithaca, NY 14853

217. Udaya B. Vemulapati, Department of Computer Science, University of Central Florida, Orlando, FL 32816-0362

218. Robert G. Voigt, National Science Foundation, Room 417, 1800 G Street N.W., Washington, DC 20550

219. Bi R. Vona, Center for Numerical Analysis, RLM 13.150, University of Texas at Austin, Austin, TX 78712

220. Henk A. van der Vorst, Professor Dept. of Mathematics, Universiteit Utrecht, P.O. Box 80010, 3508 TA, Utrecht, THE NETHERLANDS

221. Michael D. Vose, 107 Ayres Hall, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301

222. Phuong Vu, Cray Research, Inc., 19607 Franz Road, Houston, TX 77084

223. A. J. Wathen, School of Mathematics, University Walk, Bristol BS8 1TW, England

224. Robert P. Weaver, 1555 Rockmont Circle, Boulder, CO 80303

225. Mary F. Wheeler, Department of Mathematical Sciences, Rice University, P. O. Box 1892, Houston, TX 77251

226. Andrew B. White, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545

227. John Zahorjan, Department of Computer Science and Engineering, Sieg Hall, FR-35, University of Washington, Seattle, WA 98195

229–230. Office of Scientific & Technical Information, P. O. Box 62, Oak Ridge, TN 37831