**SANDIA REPORT**
SAND98–8201 • UC–411
Unlimited Release
Printed October 1997



# Parallel Unconstrained Minimization of Potential Energy in LAMMPS

Todd Plantenga



Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Approved for public release; distribution is unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.



Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

Distribution
Category UC-411

SAND98-8201
Unlimited Release
Printed October 1997

# Parallel Unconstrained Minimization of Potential Energy in LAMMPS

Todd Plantenga
Scientific Computing Department
Sandia National Laboratories
Livermore, CA

October 13, 1997

## ABSTRACT

This report describes a new minimization capability added to LAMMPS V4.0. Minimization of potential energy is used to find molecular conformations that are close to structures found in nature. The new minimization algorithm uses LAMMPS subroutines for calculating energy and force vectors, and follows the LAMMPS partitioning scheme for distributing large data objects on multiprocessor machines. Since gradient-based algorithms cannot tolerate nonsmoothness, a new Coulomb style that smoothly cuts off to zero at a finite distance is provided. This report explains the minimization algorithm and its parallel implementation within LAMMPS. Guidelines are given for invoking the algorithm and interpreting results.

## Acknowledgement

The author acknowledges support by the Department of Energy through Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

# Contents



# List of Figures



# List of Tables

Intentionally Left Blank

# Parallel Unconstrained Minimization of Potential Energy in LAMMPS

## 1 Introduction

LAMMPS [1] [1] is a parallel code developed at Sandia National Laboratories for simulating molecular dynamics. LAMMPS can model very large systems of covalently connected atoms (up to 100,000 on the Sandia Teraflop machine), including the calculation of empirical potential energy and associated force vector. This paper describes a new capability for LAMMPS Version 4.0: the determination of molecular conformations that minimize potential energy.

Some form of gradient-based energy minimization is available in most molecular modeling codes (e.g., QUANTA, AMBER, GROMOS). The goal is to find molecular structures corresponding to a *local* minimum of potential energy (there are usually many local minima). Gradient minimization is important for relaxing initial "guessed" conformations that are inadvertently at high energies, and for investigating structural changes near the global energy minimum. Potential energy minimization of very large systems is relevant to Sandia work on polymer aging, chem/bio sensor analysis and design, and protein-protein docking.

The minimization algorithm added to LAMMPS is an implicit Newton method; thus, it enjoys a fast asymptotic rate of convergence to a highly accurate solution, but requires only energy and force information. Potential energy and force calculations are made with calls to existing LAMMPS subroutines. The algorithm also requires storage and manipulation of several large vectors. These vectors are partitioned for distributed computing in the same manner as other LAMMPS vectors.

In the next section, the minimization algorithm and its parallel implementation is described in detail. §3 discusses modifications to the empirical potential energy function that make it smooth enough for gradient minimization. The final two sections show how minimization is invoked, and how it performs on massively parallel machines.

## 2 Minimization Algorithm

Gradient-based minimization methods start with an arbitrary molecular conformation, then iteratively adjust atom positions in a manner that steadily decreases the potential energy; eventually, a local energy minimum is reached. Consider a system of N atoms, and let the vector $p$ contain the 3N position coordinates of these atoms. The potential energy $E(p)$ is an algebraic function of the 3N unknowns, and $-\nabla E(p)$ is the force vector. At iteration $k$ of a minimization algorithm, atom positions are modified by adding a "step" vector $d$ to produce the next iterate $p_{k+1} = p_k + d$. Figure 1 gives a general outline of the minimization process.

The simplest algorithm, gradient descent, uses $d = -\alpha \nabla E(p_k)$, a step of length $\alpha$ in the direction of maximum energy decrease. The step moves each atom along the force vector that it experiences. Gradient descent is easy to code and always works, but is usually very slow in reaching a minimum (the asymptotic convergence rate is only linear).

[1] Contact Steve Plimpton, sjplimp@cs.sandia.gov. See http://www.cs.sandia.gov/~sjplimp/main.html.

```
start with position vector p_0
for  k = 0  to  max_iters
    if  sufficiently close to a local minimum  then stop
    calculate a step d
    if  E(p_k + d) is sufficiently less than E(p_k)
        then  p_{k+1} = p_k + d
        else  p_{k+1} = p_k,  do something that modifies calculation of d
continue
```

Figure 1: Outline of a generic minimization algorithm

More efficient minimization algorithms are based on Newton's method. They converge much faster (theoretically at a quadratic rate), but require solving linear systems involving the second partial derivatives of $E$. The Hessian matrix of $E(p)$ is defined as

$$\nabla^2 E(p) = \begin{bmatrix} \frac{\partial^2 E(p)}{\partial p_1 \partial p_1} & \cdots & \frac{\partial^2 E(p)}{\partial p_1 \partial p_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E(p)}{\partial p_N \partial p_1} & \cdots & \frac{\partial^2 E(p)}{\partial p_N \partial p_N} \end{bmatrix},$$

and the exact Newton step is given by $d = -[\nabla^2 E(p)]^{-1} \nabla E(p)$.

In LAMMPS applications the Hessian matrix is large, expensive to compute and invert, and difficult to store as a distributed object. An excellent alternative is to compute the Newton step inexactly, using a modified form of conjugate gradient. Conjugate gradient (CG) requires only Hessian matrix-vector products, which can be approximated without calculating second derivatives. This class of optimization algorithm is known as a *Hessian-free truncated Newton* method, and dates back to the work of Dembo, Eisenstat and Steihaug [2] and O'Leary [3]. It's use in molecular modeling was pioneered by Schlick and Overton [4], and is becoming more common [5, 6].

## 2.1 Hessian-free algorithm

The detailed pseudo-code for the algorithm implemented in LAMMPS is provided in Algorithms 1 and 2. The main procedure, Algorithm 1, describes the outer iteration, indexed by $k$. Algorithm 2 shows an inner CG-based iteration, indexed by $i$, that calculates an inexact "truncated" Newton step. Algorithm 2 is invoked by Algorithm 1 at (3).

Algorithm 1 encodes a *trust region* technique that adds robustness to the Newton method [7, pp. 95–107]. Newton's method is based on a two-term Taylor series expansion of the potential energy that is constructed around the conformation described by $p_k$:

$$E(p_k + d) \approx E(p_k) + d^T \nabla E(p_k) + 0.5 d^T [\nabla^2 E(p_k)] d.$$

This quadratic function is referred to as the *model* energy. Note that when the Hessian matrix is positive definite, the minimum of the model is reached by taking the exact Newton step $d = -[\nabla^2 E(p_k)]^{-1} \nabla E(p_k)$.

**Algorithm 1** *Hessian-free trust region method for energy minimization*

start with position vector $p_0$, trust radius $\Delta_0$ (1)
**for** $k = 0$ **to** max_iters
  **if** $\|\nabla E(p_k)\|_\infty \leq$ stop_tol **then stop** (2)
  calculate $d$ using HFTNCG($p = p_k, \Delta = \Delta_k, \rho = \min\{0.1/k, \|\nabla E(p_k)\|_2\}$) (3)
  compute a_red $= E(p_k) - E(p_k + d)$ (4)
  **if** $|$a_red$| < \epsilon_{mach}$ **then stop** (5)
  compute p_red $= -d^T \nabla E(p_k) - 0.5 d^T [\nabla^2 E(p_k)] d$,
    approximating $[\nabla^2 E(p_k)]d$ with (2.1) or (2.2) (6)
  **if** (p_red $> 0$ **and** a_red $> 0.1 *$ p_red) (7)
    **then** $p_{k+1} = p_k + d$, $\Delta_{k+1} \geq \Delta_k$ (8)
      repartition atoms, make new neighbor lists, recompute $E(p_{k+1})$ (9)
    **else** $p_{k+1} = p_k$, $\Delta_{k+1} < \Delta_k$ (10)
      **if** $\Delta_{k+1} < 10\epsilon_{mach}\|p_k\|_\infty$ **then stop** (11)
**continue**

**Algorithm 2** *HFTNCG subroutine for calculating d*

enter with atom positions $p$, trust radius $\Delta$, and stop tol $\rho$
$d_0 = 0$, $r_0 = -\nabla E(p)$, $t_0 = r_0$ (12)
**for** $i = 0$ **to** max_sub_iters
  approximate $w_i = [\nabla^2 E(p)]t_i$ using (2.1) or (2.2) (13)
  **if** $w_i^T t_i < \epsilon_{mach} t_i^T t_i$ (14)
    **then** find $\tau$ such that $\|d_i + \tau t_i\|_2 = \Delta$ and **return** $d \leftarrow d_i + \tau t_i$ (15)
  $\alpha_i = r_i^T r_i / t_i^T w_i$ (16)
  $d_{i+1} = d_i + \alpha_i t_i$ (17)
  **if** $\|d_{i+1}\|_2 > \Delta$ (18)
    **then** find $\tau$ such that $\|d_i + \tau t_i\|_2 = \Delta$ and **return** $d \leftarrow d_i + \tau t_i$ (19)
  $r_{i+1} = r_i - \alpha_i w_i$ (20)
  **if** $\|r_{i+1}\|_2 / \|r_0\|_2 < \rho$ (21)
    **then return** $d \leftarrow d_{i+1}$ (22)
  $\beta_i = r_{i+1}^T r_{i+1} / r_i^T r_i$ (23)
  $t_{i+1} = r_{i+1} + \beta_i t_i$ (24)
**continue**

The quadratic model is a better approximation when the step $d$ is small. Trust region methods exploit this fact by insisting that each step satisfy $\|d\|_2 \leq \Delta$, where $\Delta$ is the trust region radius (the inequality is enforced in Algorithm 2). Step (4) of Algorithm 1 computes the actual reduction (a_red) in potential energy made by $d$. Step (6) computes the "predicted" reduction (p_red) corresponding to the quadratic energy model. Step (7) determines whether the actual reduction is close enough to the predicted reduction – if so, then $d$ is accepted; otherwise, $d$ is rejected, the radius $\Delta_{k+1}$ is made smaller, and a new $d$ is calculated. Trust region methods are robust precisely because they reduce $\Delta$ when the actual and predicted reductions disagree. Eventually $\Delta$ becomes small enough that the quadratic Taylor series

expansion can be "trusted" for every step satisfying $\|d\|_2 \leq \Delta$.

Algorithm 2 computes an approximation to the Newton step using conjugate gradient. The CG subiteration can terminate at (15), (19), or (22), in each case returning a truncated Newton step $d$. Step (21) is the usual CG termination test based on reduction of the residual in the linear system being solved. The residual tolerance $\rho$ is set in step (3) of Algorithm 1. It is designed to make $d$ cheap to compute during early iterations, but accurate enough near an energy minimum to give the overall algorithm an asymptotic quadratic rate of convergence [2]. Steps (14)-(15) handle the situation in which the Hessian is not positive definite. They detect a direction of negative curvature and reduce the energy by following the direction to the edge of the trust region. Step (19) enforces the trust region inequality. Steihaug proved in [8] that the steps $d_i$ generated by Algorithm 2 are monotonically increasing in length, so it is appropriate to stop as soon as $\Delta$ is exceeded.

Steps (6) and (13) multiply the Hessian matrix by a vector. Geometrically, the product corresponds to the curvature of the energy function in the direction of the vector; mathematically, it is a type of directional derivative. To avoid forming the Hessian, the directional derivative is estimated with a finite difference formula. A forward difference approximation,

$$[\nabla^2 E(p_k)]y \approx \frac{\nabla E(p_k + \theta_f y) - \nabla E(p_k)}{\theta_f}, \tag{2.1}$$

requires one extra gradient/force evaluation. The approximation uses the safeguarded value $\theta_f = 2\sqrt{1000\epsilon_{mach}}/\|y\|_2$ [9, pp. 94–99]. When the algorithm is close to an energy minimum, a more accurate central difference approximation is used:

$$[\nabla^2 E(p_k)]y \approx \frac{\nabla E(p_k + \theta_c y) - \nabla E(p_k - \theta_c y)}{2\theta_c}, \tag{2.2}$$

with $\theta_c = (3 \times 1000\epsilon_{mach})^{(1/3)}/\|y\|_2$. The central difference estimate requires two extra gradient/force evaluations.

Algorithm 1 normally terminates when the convergence criterion at (2) is met. The user can stop it sooner by limiting the number of outer iterations, or the total number of gradient/force calculations. The algorithm also includes a safeguard at (11) that stops execution before underflow errors can occur (underflow is measured in terms of the machine precision $\epsilon_{mach}$, approximately $2.2 \times 10^{-16}$ on most machines). Step (11) provides an escape for the algorithm if the stop_tol in (2) is set too small.

## 2.2 Parallel implementation issues

LAMMPS is a data-parallel SPMD application tailored for massively parallel distributed computing architectures. LAMMPS employs a *spatial decomposition* of data objects based on the Cartesian coordinates of the atoms [10]. First, a box with sides oriented along the coordinate axes is constructed to enclose all atoms. Each side of the bounding box is subdivided (usually by a power of 2) into uniform intervals, creating a grid of equal-sized subblocks that are assigned to processors on a one-to-one basis. Each processor is assigned "ownership" of the atoms located in its subblock, including the data and force calculations associated with those atoms. Potential energy interactions extend only a limited distance, involving just the nearest neighbors of each atom. For this reason, the spatial decomposition scheme usually makes a good balance between memory needs, CPU work, and communication costs [10].

The minimization algorithm accesses atom positions and the force vector. Both of these are `DOUBLE PRECISION` arrays of up to `3*maxown` elements per processor. Algorithms 1 and 2 need storage for 8 other vectors that are used to hold intermediate results. Thus, minimization requires `8*3*maxown DOUBLE PRECISION` words on each processor. The 8 arrays are local variables, so LAMMPS memory requirements are changed only if the minimizer is invoked.

All vectors in the minimization algorithm follow the same spatial decomposition layout as the atom position and force arrays. LAMMPS subroutines are called to compute the energy and force. Additional work by the algorithm is in the form of the Level One BLAS [11] operations `daxpy`, `dcopy`, `ddot`, `dnrm2` and `dscal`. Two of these, `ddot` and `dnrm2`, require simple modifications for parallel execution. As will be seen is §5, the linear algebra costs of the algorithm are negligible compared to the costs of computing the energy and force.

When a new step is accepted in Algorithm 1, atom positions are updated at (8). This change can impact further calculations in three ways:

- neighbor lists for non-bonded interactions might change, thus altering the energy and force values,
- processors might transfer the ownership of atoms that move out of their subblock,
- the bounding box surrounding all atoms might change (if the domain is not periodic).

LAMMPS has subroutines for molecular dynamics that address these problems, and they are called at (9) in Algorithm 1. The user can control the frequency with which these subroutines are called (see §4.1), but it is recommended that repartitioning and reneighboring take place after every accepted step.

Notice that the energy $E(p_k + d)$ computed at (4) does not include reneighboring, even though the step $d$ might move atoms a significant distance. This is because minimization must evaluate $d$ with respect to the quadratic model potential defined around $p_k$. The model should not be changed until a step passes the acceptance test at (7). Then a new model is constructed around $p_{k+1}$, and minimization continues on this model. It is possible that the new $E(p_{k+1})$ computed at (9) will differ from the value $E(p_k + d)$ computed at (4), but the two values should not be compared because they correspond to two different definitions of the potential function. In practice these discrepancies are rare and do not affect convergence.

When LAMMPS repartitions and transfers atoms to new processors, it assumes that no atom moves more than one subblock from its previous location (the shift can be one subblock in each coordinate). This is a reasonable assumption for molecular dynamics because of the small time step used in integration, and it limits parallel communication costs. However, one iteration of minimization can cause much larger shifts in position, especially for a "stressed" conformation with high potential energy. To prevent an error, a check is made and the step $d$ rejected at (7) if it causes any atom to jump more than one processor. The check includes the effect of redefining the bounding box in nonperiodic problems. Rejection of the step causes $\Delta$ to decrease at (10), so that the next step $d$ will not move atoms as far.

# 3 Smoothed Potential Energy Function

The potential energy function used by LAMMPS is patterned after the standard CHARMM [12, 13] formulation. It is described parametrically by

$$\begin{aligned} E &= \sum_{bonds} k_b(r_{ij} - r_{ij}^0)^2 + \sum_{angles} k_a(\theta_{ijk} - \theta_{ijk}^0)^2 \\ &+ \sum_{dihedrals} E_{dih}(\phi_{ijkl}) + \sum_{impropers} E_{imp}(\xi_{ijkl}) \\ &+ \sum_{pairs\ i,j} \omega_{coul}(r_{ij}) k_{coul} \frac{q_i q_j}{r_{ij}} + \omega_{L\text{-}J}(r_{ij})\, 4\epsilon_{ij} \left[ \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] \end{aligned} \tag{3.1}$$

(LAMMPS also supplies lumped approximations for long range forces, but these are not considered here). The terms comprising $E$ model, respectively, covalent bond lengths, valence angles, dihedral rotation barriers between four atoms, improper angles (out-of-plane motions) between four atoms, and non-bonded coulombic and Lennard-Jones potentials (multiplied by cutoff functions $\omega_{coul}$ and $\omega_{L\text{-}J}$, respectively). The variables of interest are $r_{ij}$, the Euclidean distance between atoms $i$ and $j$. Only $r_{ij}$, $\theta_{ijk}$, $\phi_{ijkl}$, and $\xi_{ijkl}$ depend on atom positions; all other quantities are constants.

Newton-based minimization requires that the function being minimized have continuous second derivatives. The potential energy function used in LAMMPS meets this criteria, except for the cutoff options. Both the coulomb and Lennard-Jones interactions typically use a sharp cutoff function defined by the rules

$$\begin{aligned} &\textbf{if}\ \ r < r_{cut}\ \ \textbf{then}\ \ \omega(r) = 1 \\ &\textbf{if}\ \ r \geq r_{cut}\ \ \textbf{then}\ \ \omega(r) = 0\ . \end{aligned}$$

This function is not continuous, and causes a significant jump in the magnitude of the coulomb potential unless $r_{cut}$ is very large (for $k_{coul} = 332.0636, q_i = q_j = 0.5$, the discontinuity at 20Å is 4.15 kcal/mol). The jump creates insurmountable problems for a gradient-based minimization algorithm, causing it to halt prematurely. The reason for this is not too hard to explain.

Suppose a pair of atoms with charges of like sign become situated during minimization with interatomic distance $r_{ij}$ slightly greater than $r_{cut}$. The atoms are considered "neighbors" and a repulsive coulomb force exists, but is truncated to zero by $\omega_{coul}$. If other forces conspire to push the atoms closer together, then the trial step $d$ generated by Algorithm 2 will change the atom positions so that their distance is less than $r_{cut}$. Step (4) suddenly sees the repulsive force when computing $E(p_k + d)$, but it is still truncated to zero in the quadratic model employed in (6). The step is therefore rejected at (7), the trust radius is decreased, and the algorithm seeks a new, shorter, step $d$ with the same quadratic model. Iterations continue in this manner until $\Delta$ is so close to zero that the algorithm halts at (11). No step can be accepted because the smooth quadratic model cannot represent the nonsmooth coulomb cutoff function. And it is easy to see that the "unlucky" circumstance of two repulsive atoms becoming situated with $r_{ij}$ just slightly greater than $r_{cut}$ is not coincidental. When $r_{ij}$ is larger, steps that cross the discontinuity are rejected, but a shorter step $p_k + d$ that does not decrease $r_{ij}$ below $r_{cut}$ is accepted. The algorithm therefore accepts a series of small steps that nudge the atoms closer and closer towards an interatomic distance of $r_{cut}$.

To make minimization possible, the new release of LAMMPS includes the smooth coulomb cutoff function used in CCEMD [14]. This cutoff is a cubic polynomial fitted between the cutoff distance $r_{cut}$ and a smaller threshold distance $r_{inner}$:

$$\begin{aligned}
&\textbf{if} \quad r < r_{inner} \quad &&\textbf{then} \quad \omega_{coul}(r) = 1 \\
&\textbf{if} \quad r_{inner} \leq r < r_{inner} \quad &&\textbf{then} \quad \omega_{coul}(r) = \frac{(r_{cut} - r)^2(r_{cut} + 2r - 3r_{inner})}{(r_{cut} - r_{inner})^3} \\
&\textbf{if} \quad r \geq r_{cut} \quad &&\textbf{then} \quad \omega_{coul}(r) = 0.
\end{aligned}$$

This function has continuous second derivatives, and has the virtue of leaving coulomb potential terms unchanged when $r < r_{inner}$. It executes about 10% slower than the simpler sharp cutoff function.

A corresponding smooth cubic cutoff function for Lennard-Jones potentials was not implemented. LAMMPS Version 4.0 currently supplies the sharp cutoff function described earlier, and a "switched" cutoff option defined by

$$\begin{aligned}
&\textbf{if} \quad r < r_{inner} \quad &&\textbf{then} \quad E_{L\text{-}J}(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] - b \\
&\textbf{if} \quad r_{inner} \leq r < r_{cut} \quad &&\textbf{then} \quad E_{L\text{-}J}(r) = a_0 - a_1(r - r_{inner}) - a_2(r - r_{inner})^2/2 \\
&&&\qquad\qquad -a_3(r - r_{inner})^3/3 - a_4(r - r_{inner})^4/4 - b \\
&\textbf{if} \quad r \geq r_{cut} \quad &&\textbf{then} \quad E_{L\text{-}J}(r) = 0,
\end{aligned}$$

where

$$\begin{aligned}
a_0 &= 4\epsilon\left[\left(\frac{\sigma}{r_{inner}}\right)^{12} - \left(\frac{\sigma}{r_{inner}}\right)^{6}\right] \\
a_1 &= \frac{\epsilon}{r_{inner}}\left[48\left(\frac{\sigma}{r_{inner}}\right)^{12} - 24\left(\frac{\sigma}{r_{inner}}\right)^{6}\right] \\
a_2 &= -\frac{\epsilon}{r_{inner}^2}\left[13 \times 48\left(\frac{\sigma}{r_{inner}}\right)^{12} - 7 \times 24\left(\frac{\sigma}{r_{inner}}\right)^{6}\right] \\
a_3 &= -\frac{3}{(r_{cut} - r_{inner})^2}\left[a_1 + \frac{2}{3}(r_{cut} - r_{inner})a_2\right] \\
a_4 &= -\frac{1}{3(r_{cut} - r_{inner})^2}\left[a_2 + 2(r_{cut} - r_{inner})a_3\right] \\
b &= a_0 - a_1(r_{cut} - r_{inner}) - a_2(r_{cut} - r_{inner})^2/2 \\
&\quad -a_3(r_{cut} - r_{inner})^3/3 - a_4(r_{cut} - r_{inner})^4/4.
\end{aligned}$$

The switched Lennard-Jones cutoff is smooth and goes to zero for $r \geq r_{cut}$; however, it does not equal the standard Lennard-Jones potential for $r < r_{inner}$.

The sharply cutoff Lennard-Jones potential usually has only a small discontinuity. For example, the Lennard-Jones interaction between two alpha carbons is characterized by $\epsilon = 0.0498, \sigma = 2.265$. The discontinuity at 10Å is $2.69 \times 10^{-5}$ kcal/mol, and the jump at 20Å is only $4.20 \times 10^{-7}$ kcal/mol. Unless minimization is carried to extremely high accuracy, these minute discontinuities will not cause difficulties. Thus, the Lennard-Jones cutoff style can be used, provided that $r_{cut}$ is not too small.

# 4 Using the Software

## 4.1 Inputs

LAMMPS uses a text file of control commands to set up parameters and direct execution. Three new commands are defined for the minimization algorithm:

```
min file        filename
min style       hftn
minimize        stop_tol   max_iters   max_force_calculations
```

The first command specifies a filename for the minimization output. Any existing file is deleted as soon as minimization begins. If the line is omitted, no output file will be created.

The second command selects a minimization algorithm. Currently only `hftn` is defined, but other algorithms may be implemented in the future.

The last command begins execution of the minimizer. Its three parameters determine how the algorithm stops. The parameter *stop_tol* maps directly to step (2) of Algorithm 1, and stops execution as soon as the magnitude of every force vector component is less than or equal to it. The parameter *max_iters* caps the number of outer iterations made by Algorithm 1. The third parameter limits the number of force calculations made by the algorithm. It provides the best way to control execution time, because force calculations are the dominant cost in large problems. The algorithm may make one or two force calculations beyond *max_force_calculations* if it is in the middle of a finite difference approximation. Default values for these three parameters are 0.0001, 100, and 1000. Since this command starts the algorithm, the first two commands should precede it.

A few other LAMMPS input commands have an effect on the minimization algorithm. As described in §3, the coulomb force field option must be

```
coulomb style   coul/smooth  inner   outer
```

Two of the parameters set by the "neighbor" command also have an effect:

```
neighbor        skin   neighstyle   neighfreq   neighdelay   neightrigger
```

The *skin* parameter implicitly limits step sizes by causing any step which moves an atom more than 1/2 the skin depth to be rejected. If *skin* is zero, atoms can move up to 5Å. The *neighfreq* parameter controls how many accepted steps elapse between the repartitioning and reneighboring calls made at (9) in Algorithm 1. The recommended value is one (reneighbor after every step), since the costs are small compared to the large number of force calculations made to compute each step $d$.

## 4.2 Outputs

Figure 2 shows a portion of the output created by the minimizer. At the top of the output the stopping criteria, force field styles, and other input control parameters are echoed. Then comes a detailed breakdown of the potential energy at the start of minimization – a similar breakdown is printed at the bottom of the output for the final conformation generated by the algorithm. These two printouts should match the LAMMPS log file output exactly.

```
Find local energy minimum using truncated Newton CG
  Stop minimization if:
       ||grad||_inf .LE. 1.000E-05            OR
       max number force calculations >  50000 OR
       max number iterations         >   1000
  Max atom displacement per step is  5.000 Ang
  Exchange & reneighbor after      1 accepted steps

Coulomb forces smooth, cut off between    10.00 and    20.00 Angstroms
L-J     forces sharply cut off at         20.00 Angstroms

====================================================================
  Total PE  =     -1263.6073
  E (bond)  =         9.0474          E (coulomb) =     -1353.2352
  E (angle) =        25.0634          E (VDW)     =       -19.7982
  E (dih)   =        71.1870          E (long)    =         0.0000
  E (imprp) =         4.1283
====================================================================

Truncated Newton CG with trust regions on   1365 unknowns

     Iter      f(x)     ||grad||_inf   Delta     ||step||  f evals     ared         pred       CG iters
    -----  -----------  ----------  ---------  ---------  -------  ----------  ----------    --------
        0  -1.2636E+03  7.2020E+01  3.695E+00                  1
   <forward diffs>
        1  -1.4801E+03  1.5901E+02  7.389E+00  3.695E+00       12   2.165E+02   2.394E+02       9  TR
   (       -1.4800E+03  1.5898E+02                            13      after reneighboring )
        2  -1.5786E+03  1.6475E+02  7.389E+00  7.389E+00       28   9.856E+01   2.993E+02      13  TR
   (       -1.5787E+03  1.6475E+02                            29      after reneighboring )
rej     3  -1.2944E+03               3.010E+00  7.389E+00       48  -2.844E+02   3.959E+02      17  TR
        4  -1.8784E+03  1.8260E+02  6.021E+00  3.010E+00       63   2.996E+02   3.039E+02      13  TR
   (       -1.8783E+03  1.8262E+02                            64      after reneighboring )
        .                                                       .
        .                                                       .
      273  -2.8846E+03  6.3343E-02  1.474E+00  7.372E-01    24480   1.966E-02   2.003E-02     376  TR
   (       -2.8846E+03  6.3343E-02                           24481      after reneighboring )
      274  -2.8846E+03  9.6155E-03  1.474E+00  1.308E-01    25698   5.458E-04   5.445E-04    1215  Nw
   (       -2.8846E+03  1.0489E-02                           25699      after reneighboring )
      275  -2.8846E+03  1.9118E-05  1.474E+00  4.134E-03    26767   1.098E-06   1.022E-06    1066  Nw
   (       -2.8846E+03  1.9118E-05                           26768      after reneighboring )
   <central diffs>
      276  -2.8846E+03  1.9220E-07  1.474E+00  4.015E-05    30257   4.129E-10   4.091E-11    1743  Nw
   (       -2.8846E+03  1.9220E-07                           30258      after reneighboring )
 +++ ||g||_inf less than tolerance

  Number of force calcs                       =    30258
  Number of force calcs without non-bonded E =    29777
  Number of repartitionings                   =      205

CPU time for minimization, per proc (secs):
  (raw) avg =    7630.21  (per f&g eval) avg =   0.2522
        max =    7664.60
CPU time in Compute_f_g, total:
  (raw) avg =    7393.13  (per f&g eval) avg =   0.2443
        max =    7434.39
CPU time in Compute_f_g, just energy:
  (raw) avg =      88.97  (per f&g eval) avg =   0.0029
        max =      89.29
CPU time in Compute_f_g, lost due to load imbalance:
  (raw) avg =    3744.59  (per f&g eval) avg =   0.1238
        max =    5619.92

CPU times from LAMMPS, per proc (secs), % of    7630.2050
          Avg Time      %      Max Time      %
         ------------  -----  ------------  -----
Bond          20.5361   0.27       40.0022   0.52
Angle         67.4046   0.88      135.0166   1.77
Dihedral     127.4567   1.67      260.9897   3.42
Improper      29.3723   0.38       60.1945   0.79
Nonbond     3268.8508  42.84     6785.8693  88.93
Long           0.8139   0.01        0.8210   0.01
Neighbor      25.4444   0.33       46.3401   0.61
Chk neigh      0.3487   0.00        0.6540   0.01
Exchange       0.4386   0.01        0.5142   0.01
Comm          84.1061   1.10       92.3113   1.21
Rev Comm      41.0251   0.54       50.9747   0.67
IO            15.3540   0.20       17.0566   0.22
(other)     3949.0538  51.76

====================================================================
  Total PE  =     -2884.5698
  E (bond)  =        12.3621          E (coulomb) =     -3029.2057
  E (angle) =        72.1164          E (VDW)     =       -68.1287
  E (dih)   =       122.8032          E (long)    =         0.0000
  E (imprp) =         5.4828
====================================================================
```

Figure 2: Sample output from the minimization algorithm

The middle of the output contains a long table with nine columns of information. Each numbered line in the table gives a summary of one minimization iteration. If the iteration resulted in a step that was rejected, then the letters `rej` appear at the beginning of the line; otherwise, it may be assumed the trial step was accepted. Data in the nine columns refer to values at the end of an iteration, except for the trust region radius. The nine columns of data are:

| | |
|---|---|
| `Iter` | Iteration number $k$ in Algorithm 1 |
| `f(x)` | Energy at the end of the iteration (units here are kcal/mol) |
| `||grad||_inf` | Largest magnitude of a force component (i.e., $\|\nabla E(p)\|_\infty$) |
| `Delta` | Trust radius $\Delta$ at the start of the iteration<br>(note all other data refers to the end of the iteration) |
| `||step||` | Length of the step $d$ in Angstroms (i.e., $\|d\|_2$) |
| `f evals` | Cumulative number of force evaluations made so far |
| `ared` | Actual reduction in energy made by $d$ (see (4) in Algorithm 1) |
| `pred` | Predicted reduction in energy (see (6) in Algorithm 1) |
| `CG iters` | Number of inner iterations in Algorithm 2, and why it exited:<br>`TR` - exceeded the trust radius at (19)<br>`Nw` - converged to a Newton step at (22)<br>`ng` - encountered a direction of negative curvature at (15)<br>`it` - reached max_sub_iters in Algorithm 2<br>`FD` - approximation (2.1) was inadequate in (13) |

After each accepted step a line in parentheses shows the new energy and gradient/force norm computed after repartitioning and reneighboring. For example, the energy after iterations 1, 2, and 4 changed slightly as a result of reneighboring, but iterations 275 and 276 did not (step sizes near a solution are usually very small). Before iteration 276 the message `<central diffs>` indicates that the algorithm switched from equation (2.1) to (2.2) for greater accuracy.

At the end of the table in Figure 2 is a message beginning with `+++` that tells why the minimizer stopped – in this example because it converged. Then come three lines of summarizing totals. Next are a group of four CPU times. The cumulative `raw` time, averaged over all processors, is listed, followed by the same time divided by the number of force calculations (`per f&g eval`). This last average is the best way to assess scalability of the software, since the exact number of minimization iterations is sensitive to the number of processors (roundoff errors change the inner iteration at which Algorithm 2 exits). The first of the four CPU times gives the total time of LAMMPS execution. The second measures the time spent computing energy and force vectors. The third and fourth numbers break the second number down into more detail. Procedure `Compute_f_g` first uses LAMMPS subroutines to compute local force vector components in parallel, then swaps information between processors to get the total force, then makes an additional calculation of non-bonded potential energy. Usually the minimization algorithm only wants the force, so the energy calculation is skipped. The third CPU time gives the cost of the additional energy computations, averaged over all force calculations (which is admittedly a little misleading). The fourth CPU time is measured between the end of the local force calculations and the start of synchronous interprocessor force vector communications; thus, it gives time lost due to work load imbalances. Of course there are other places where time is lost due to load imbalance, but this is by far the most significant.

Finally, Figure 2 contains a list of CPU times and percentages from LAMMPS timers. The second and third columns are the most useful. They show the CPU time spent by an

average processor on each task. The other time is largely accounted for by adding the times for Compute_f_g, just energy and Compute_f_g, lost due to load imbalance. In this example, those tasks take $88.97 + 3744.59 = 3833.56$ of the 3949.05 seconds in other. The rest of other is largely due to linear algebra computations in the minimization algorithm.

### 4.3 Software structure

Algorithms 1 and 2 are coded in the Fortran source file min_algs.f. Supporting subroutines that interface it to LAMMPS are in the file min_support.f. The minimization algorithm requires the following standard BLAS [11] routines, which can be obtained from netlib@ornl.gov or http://www.netlib.org: daxpy.f, dcopy.f, ddot.f, dnrm2.f, and dscal.f. Appendix A provides a table of source files in the new release of LAMMPS.

A small number of global variables were added to the LAMMPS set defined in the file lammps.h. The source code for these additions is given below.

```
CHARACTER*80      opt_outfile
INTEGER           opt_algthm, opt_max_iters, opt_max_fns
REAL*8            opt_stop_tol
DOUBLE PRECISION  opt_time1, opt_time2, opt_time3

COMMON /bk150/ opt_outfile
COMMON /bk151/ opt_algthm, opt_max_iters, opt_max_fns
COMMON /bk152/ opt_stop_tol
COMMON /bk153/ opt_time1, opt_time2, opt_time3
```

## 5 Performance of the Minimization Algorithm

Newton-based minimization methods have a characteristic convergence behavior on molecular structures with covalent bonds. Figure 3 plots the potential energy $E$ and force magnitude $\|\nabla E\|_\infty$ against execution time for a test molecule (Transcription Regulation Protein 1ROP [15], consisting of 455 atoms). Only data for accepted steps are plotted, so the energy decreases monotonically. Note that the lower graph plots the log of $\|\nabla E\|_\infty$; thus, at the final conformation all force vector components have magnitudes smaller than $2 \times 10^{-7}$ kcal/mol-Å.

Figure 3 shows that most of the energy decrease occurred in the first 1000 seconds of execution – the last 7000 seconds reduced the energy only 63 kcal/mol. Algorithm behavior from 2000 seconds to 4000 seconds was particularly vexing. Here the potential energy decreased only 1.1 kcal/mol, yet the force magnitude indicated the conformation was still far from a local minimum. This poor performance is due to ill-conditioning in the potential energy function, which can be traced primarily to the covalent bond terms [4, 16]. Ill-conditioning can be reduced with special preconditioners [4, 5, 17] or distance constraints [16, 18].

The minimization algorithm has been tested on the Sandia Teraflop machine, Intel Paragon, and a Silicon Graphics Power Challenge. Tests indicate that 95% of CPU time is spent in subroutine Compute_f_g, which uses LAMMPS routines to calculate force vectors and potential energy. Thus, the linear algebra of minimization accounts for less than 5% of CPU time, and the parallel scalability of minimization should be quite similar to the scalability of molecular dynamics simulations. Minimization requires use of the new coul/smooth force field, which appears to run about 10% slower than the coul/cut option.

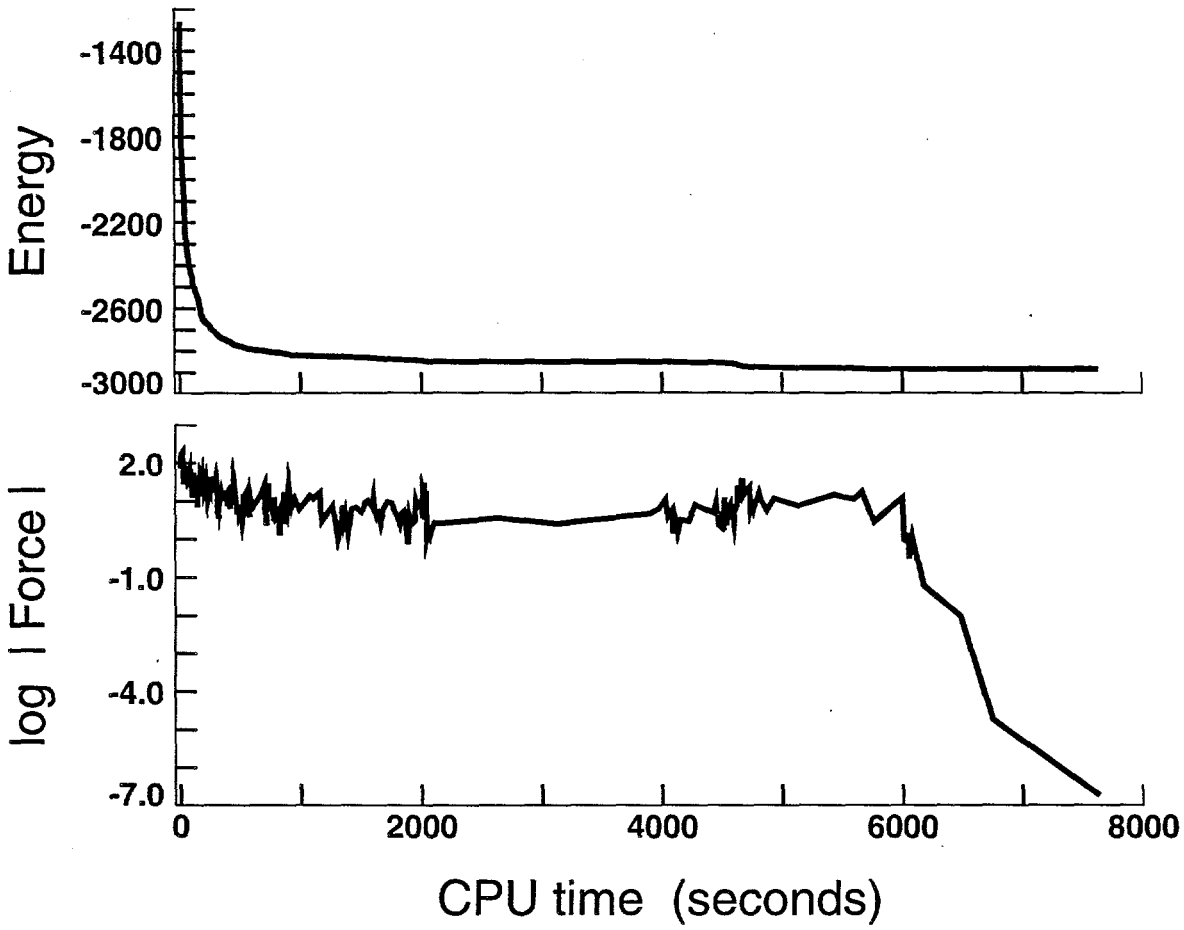


Figure 3: Minimization performance on a protein containing 455 atoms

The upper graph shows the decrease in potential energy (in kcal/mol) as a function of execution time. The lower graph plots the corresponding change in the logarithm of $\|\nabla E\|_\infty$. Atoms experience forces of more than one kcal/mol-Å for the first 6000 seconds of minimization, even though the energy barely decreases after 2000 seconds. The slow convergence in this region is due to ill-conditioning. Final convergence (beyond the 6000 second mark) is at a quadratic rate.

The minimization algorithm did not always obtain good performance when scaled to a large number of processors. Subroutine `Compute_f_g`, where 95% of CPU time is spent, incurs some losses due to processor load imbalance. Each processor computes force terms on the atoms it "owns", then synchronously distributes the results to neighboring processors. The time delay between the last local force term calculation and receipt of the first communication message is idle time. It is reported in the output as `CPU time in Compute_f_g, lost due to load imbalance` (near the bottom in Figure 2). The time loss is larger in processors that are lightly loaded.

Tables I and II show algorithm performance for different numbers of processors. Table I results from minimization of the 1ROP protein on the Teraflop machine. Table II contains data from the Paragon for a periodic block of cross-linked polymer strands (sulfur cured EPDM

rubber [2]) containing 10,954 atoms. If perfect scaling were achieved on these tests, then the *total* time (2nd column) would decrease in proportion to the number of processors. The 5th column indicates that some speedup loss is due to interprocessor communication costs. However, much greater loss is due to work load imbalance of the force calculations, as shown by the 4th column. The work load becomes more unbalanced because the spatial decomposition reflects fine grain inhomogeneities in the molecule's geometry. The *imbalance ratios* (last two columns) measure mismatch in the work load. For example, a perfectly balanced distribution of 455 atoms over 64 processors would put 7 or 8 atoms in each processor; in practice, the geometry of 1ROP concentrates 29 atoms in one processor (giving a ratio of $(455/64)/29 = 0.25$) and leaves zero atoms in several others.

Table I: Algorithm performance – 455 atoms

| | *Time per eval, in msec* | | | *Communication* | *Imbalance ratios* | |
|---|---|---|---|---|---|---|
| *# procs* | *total* | *E and force* | *imbalance* | *costs* | *local # atoms* | *local # neighbors* |
| 1 | 123.6 | 120.1 | 0.0 | 0.0 % | 1.00 | 1.00 |
| 2 | 75.1 | 71.0 | 10.9 | 1.2 % | 0.86 | 0.83 |
| 4 | 65.8 | 62.0 | 30.9 | 2.2 % | 0.51 | 0.47 |
| 8 | 40.1 | 34.7 | 19.0 | 4.8 % | 0.44 | 0.45 |
| 16 | 33.3 | 28.6 | 20.0 | 7.1 % | 0.31 | 0.26 |
| 32 | 20.9 | 16.4 | 11.6 | 12.2 % | 0.28 | 0.26 |
| 64 | 16.3 | 11.4 | 8.4 | 18.2 % | 0.25 | 0.19 |

Table II: Algorithm performance – 10,954 atoms

| | *Time per eval, in msec* | | | *Communication* | *Imbalance ratios* | |
|---|---|---|---|---|---|---|
| *# procs* | *total* | *E and force* | *imbalance* | *costs* | *local # atoms* | *local # neighbors* |
| 128 | 1731.2 | 1512.4 | 438.5 | 3.5 % | 0.75 | 0.70 |
| 256 | 927.1 | 800.2 | 258.5 | 5.8 % | 0.68 | 0.63 |
| 512 | 554.9 | 467.2 | 178.6 | 9.0 % | 0.58 | 0.54 |
| 1024 | 353.6 | 288.2 | 131.0 | 13.7 % | 0.49 | 0.51 |

Each line in a table shows the performance for a different number of processors (*# procs*). All values are the average for one processor. Columns under *Time per eval* are further averaged over the number of calls to subroutine `Compute_f_g`. The three columns give the average time spent in one call to the subroutine (*total*), average time for one force/energy computation (*E and force*), and the average time wasted in a force computation due to load imbalance (*imbalance*). *Communication costs* are a percentage of CPU time measured by adding the LAMMPS times labeled `Exchange`, `Comm`, and `Rev Comm` near the bottom of Figure 2. *Imbalance ratios* are derived from LAMMPS summary statistics. The two columns give the average number of atoms owned by a processor divided by the maximum number owned by the most heavily loaded processor (*local # atoms*), and a similar ratio involving the number of neighboring atoms stored by a processor (*local # neighbors*).

Table III and Figure 4 summarize the speedup analysis for the 455 atom example. Force computation imbalance and interprocessor communication costs account for most of the performance decrease observed in this problem. The last column of Table III implies that some other loss mechanism becomes significant for large numbers of processors, probably a load imbalance in Level One BLAS linear algebra operations. Practically, one may conclude it doesn't make much sense to run this small problem on more than 4 or 8 processors.

[2]Supplied by J.-L. Faulon, Sandia National Laboratories.

Table III: Speedups – 455 atoms

| *# procs* | *Speedup (ideal)* | *Speedup after imbalance removed* | *Speedup after imbalance and comm removed* |
|---|---|---|---|
| 1 | – (1) | – | – |
| 2 | 1.65 (2) | 1.95 | 1.98 |
| 4 | 1.88 (4) | 3.84 | 3.92 |
| 8 | 3.08 (8) | 6.92 | 7.28 |
| 16 | 3.71 (16) | 13.0 | 14.0 |
| 32 | 5.91 (32) | 21.9 | 25.0 |
| 64 | 7.58 (64) | 34.5 | 42.1 |

Data is derived from Table I. *Speedup* is the *total* time for one processor (123.6 msec) divided by the *total* time for a larger number of processors; ideally, it equals the number of processors used. This speedup is divided by the average of the two *imbalance ratios* in Table I to estimate the *speedup after imbalance removed.* The last column estimates the speedup if communication costs are also factored out. Example calculation: for 2 processors, speedup is $123.6/75.1 = 1.65$, after removing load imbalance it is $1.65/0.845 = 1.95$, and after further removing communication costs it is $1.95/(1.000 - 0.012) = 1.98$.

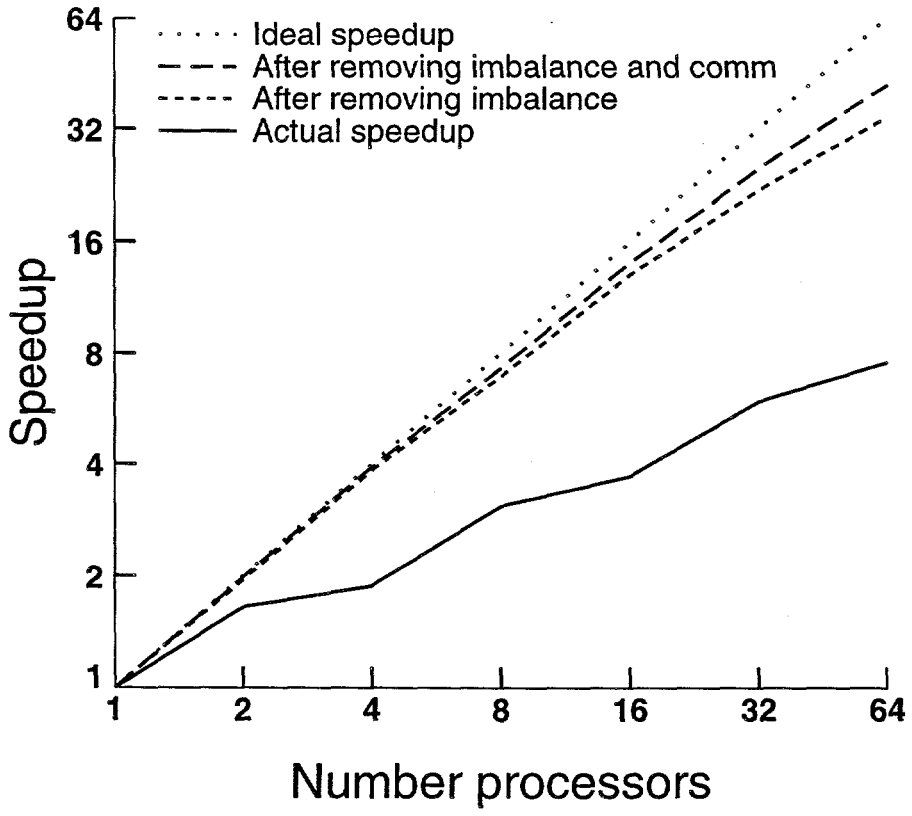


Figure 4: Speedups plotted from Table III

Table IV shows speedup for the 10,954 atom polymer. This molecular system is one piece of a much larger polymer structure, so it should be more homogeneous; nevertheless, for a sufficiently large number of processors, local geometric inhomogeneities again cause significant work load imbalance.

Table IV: Speedup – 10,954 atoms

| *# procs* | *Speedup (ideal)* | |
|---|---|---|
| 128 | – | (1) |
| 256 | 1.87 | (2) |
| 512 | 3.12 | (4) |
| 1024 | 4.90 | (8) |

# 6 Conclusion

A gradient-based minimization algorithm has been added to LAMMPS V4.0 to enable calculation of molecular conformations with minimal potential energy. The algorithm is a Newton method, but uses conjugate gradient and finite difference approximations to avoid constructing the Hessian matrix of second derivatives. The parallel implementation uses distributed vectors that follow the LAMMPS spatial decomposition scheme, and linear algebra operations performed by parallelized Level One BLAS routines. A smooth coulomb cutoff function based on cubic spline interpolation has been added to overcome pathological minimization behavior. Testing shows that the algorithm has an asymptotic quadratic rate of convergence, but progresses slowly during intermediate stages of minimization because of ill-conditioning in the potential energy function. Nearly all CPU time is spent computing force vectors. This calculation became significantly unbalanced for both test problems when the number of processors increased too much.

# References


[1] S. J. Plimpton, R. Pollock, and M. Stevens. Particle-mesh Ewald and rRESPA for parallel molecular dynamics simulations. In *Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, MN, March 1997.

[2] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.

[3] D. P. O'Leary. A discrete Newton algorithm for minimizing a function of many variables. *Math. Prog.*, 23:20–33, 1982.

[4] T. Schlick and M. Overton. A powerful truncated Newton method for potential energy minimization. *J. Comp. Chem.*, 8:1025–1039, 1987.

[5] P. Derreumaux, G. Zhang, and T. Schlick. A truncated Newton minimizer adapted for CHARMm and biomolecular applications. *J. Comp. Chem.*, 15:532–552, 1994.

[6] D. A. Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham III, S. DeBolt, D. Ferguson, G. Seibel, and P. Kollman. AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. *Computer Physics Communications*, 91:1–41, 1995.

[7] R. Fletcher. *Practical Methods of Optimization.* Wiley & Sons, Chichester, UK, second edition, 1990.

[8] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM J. Numer. Anal.*, 20:626–637, 1983.

[9] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, N.J., 1983.

[10] S. J. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.*, 117:1–19, 1995.

[11] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostronchov, and D. Sorensen. *LAPACK User's Guide*. SIAM, Philadelphia, 1992.

[12] B.R. Brooks, R.E. Bruccoleri, B.D. Olafson, D.J. States, S. Swaminatha, and M. Karplus. CHARMm: A program for macromolecular energy, minimization, and dynamics calculations. *J. Comp. Chem.*, 4:187–217, 1983.

[13] Lennart Nilsson and Martin Karplus. Empirical energy functions for energy minimization and dynamics of nucleic acids. *J. Comp. Chem.*, 7:591–616, 1986.

[14] R. Judson, D. Barsky, T. Faulkner, D. McGarrah, C. Melius, J. Meza, E. Mori, T. Plantenga, and A. Windemuth. CCEMD - Center for Computational Engineering Molecular Dynamics: Theory and user's guide, version 2.2. Technical Report SAND95-8258, Sandia National Laboratories, Livermore, CA, 1995.

[15] D. W. Banner, M. Kokkinidis, and D. Tsernoglou. Structure of the col*E1 ROP protein at 1.7 angstroms resolution. *J. Mol. Biol.*, 196:657, 1987.

[16] T. D. Plantenga and R. S. Judson. Energy minimization along dihedrals in Cartesian coordinates using constrained optimization. Technical Report SAND95-8724, Sandia National Laboratories, Livermore, CA, 1995.

[17] Tamar Schlick. Optimization methods in computational chemistry. In K. B. Lipkowitz and D. B. Boyd, editors, *Reviews in Computational Chemistry*, volume 3, pages 1–72. VCH Publishers, New York, 1992.

[18] T. D. Plantenga and R. S. Judson. Fast energy minimization of large polymers using constrained optimization. In progress, 1997.

# Appendix A LAMMPS Source File Status

| | |
|---|---|
| communicate.f | unchanged |
| daxpy.f | new file |
| dcopy.f | new file |
| ddot.f | new file |
| diagnostic.f | unchanged |
| dnrm2.f | new file |
| dscal.f | new file |
| ewald.f | unchanged |
| ewald_coeff.f | unchanged |
| finish.f | modified for smooth coulomb |
| fix.f | unchanged |
| force.f | modified for smooth coulomb |
| force_bond.f | unchanged |
| force_class2.f | unchanged |
| force_many.f | unchanged |
| force_respa.f | modified for smooth coulomb |
| initialize.f | modified for minimization |
| input.f | modified for minimization and smooth coulomb |
| integrate.f | modified for smooth coulomb |
| integrate_respa.f | unchanged |
| lammps.f | modified for smooth coulomb |
| lammps.h | modified for minimization and smooth coulomb |
| min_algs.f | new file |
| min_support.f | new file |
| misc.f | unchanged |
| neighbor.f | unchanged |
| output.f | unchanged |
| param.h | unchanged |
| parlib_unix.f | unchanged |
| pppm.f | unchanged |
| pppm_coeff.f | unchanged |
| pppm2.f | unchanged |
| pppm2_coeff.f | unchanged |
| pppm2_remap.f | unchanged |
| random.f | unchanged |
| read_data.f | unchanged |
| read_restart.f | unchanged |
| setup.f | unchanged |
| setup_special.f | unchanged |
| start.f | modified for smooth coulomb |
| string.f | unchanged |
| stringlib_unix.f | unchanged |
| thermo.f | modified for smooth coulomb |
| velocity.f | unchanged |

## DISTRIBUTION:

| | | |
|---|---|---|
| 1 | MS 9001 | Thomas Hunter, 8000<br>Attn: J. B. Wright, 2200<br>J. F. Ney (A), 5200<br>M. E. John, 8100<br>W. J. McLean, 8300<br>R. C. Wayne, 8400<br>P. N. Smith, 8500<br>P. E. Brewer, 8600<br>T. M. Dyer, 8700<br>L. A. Hiles, 8800<br>D. L. Crawford, 8900 |
| 1 | MS 9012 | Juan C. Meza, 8950 |
| 1 | MS 9214 | L. M. Napolitano, Jr., 8130 |
| 20 | MS 9214 | Todd Plantenga, 8950 |
| 3 | MS 9018 | Central Technical Files, 8940-2 |
| 4 | MS 0899 | Technical Library, 4916 |
| 1 | MS 9021 | Technical Communications Department, 8815/Technical Library, MS 0899, 4916 |
| 2 | MS 9021 | Technical Communications Department, 8815 for DOE/OSTI |