Performance Models on QCDOC for Molecular Dynamics with Coulomb Potentials[1]

Y. Deng[2] and J. Glimm
Department of Applied Mathematics, Stony Brook University, Stony Brook, NY 11794 and
Center for Data Intensive Computing, Brookhaven National Laboratory, Upton, NY 11973

J. W. Davenport
Center for Data Intensive Computing, Brookhaven National Laboratory, Upton, NY 11973

X. Cai
University of Science and Technology of China, Hefei, China 230026

E. Santos
Department of Computer Science, Virginia Tech., Blacksburg, VA 24061

Submitted to *Int. J. High Performance Computing Applications* on January 17, 2003
Submitted to *Int. J. High Performance Computing Applications* on June 30, 2003 (revised)

Abstract

We estimate that a novel architecture massively parallel computer, the QCDOC, can integrate molecular dynamics equations for 10^5 particles interacting via long range forces (including Coulomb) for one to ten microseconds of simulated time using several weeks of computing time using 8,000 or 10,000 processors. This number of atoms is typical for biological molecules. The two main conclusions we reach are:

1. This is an increase of more than one order of magnitude in simulated time over current simulations.
2. The novel architecture, with 24 parallel channels of low latency communication per processor, allows improved long range communication and an unusual degree of fine scale parallelism, as compared to conventional switch-based architectures.

The technical heart of the paper is a detailed analysis of the computing time used in the Ewald method as a function of the required accuracy, the size of the molecular dynamics cell, and the hardware design parameters.

[1] Supported in part by DOE contracts DE-FG02-90ER25084 and DE-AC02-98CH10886, the National Science Foundation DMS-0102480 and the Army Research Office grant DAAG-01-0642.

[2] Corresponding author's email address: Yuefan.Deng@StonyBrook.edu.

1. Introduction

We explore here the use of a machine with novel architecture designed for quantum chromodynamics (QCD) by a team of computer scientists and elementary particle physicists [1] mainly at Columbia University, The Riken BNL Research Center and IBM. We examine the possible use of this machine for a class of problems of general importance. This machine consists of approximately 10,000 IBM PowerPC processors with extremely fast nearest neighbor communication and 4 megabytes of memory located on the each chip. For this reason it has been named QCDOC for quantum chromodynamics on a chip. It is expected to achieve on the order of 10 teraflops peak (5 teraflops sustained) at a cost of $1 per sustained Mflop.

The closest performance analogue to this machine in common use in recent times is the Thinking Machine Corporation's CM5, because in the timing estimates for several basic algorithms, message latency can be neglected and the memory behaves as if it were shared. A programming model for the QCDOC is close to that used for standard distributed memory machines, with the caveat that message management (as well as memory management) is required of the programmer. A related machine, Blue Gene/L, is under development at IBM [2].

Our main conclusion is that the QCDOC high performance network (high bandwidth due to multiple channels per processor and the low latency) allows a very high level of parallelism (small problem size per processor), and very fast solutions, for a number of basic algorithms of wide interest. Because of the high level of parallelism, small problem sizes per processor are feasible, with the result that the entire problem fits in the on chip (L2 cache) memory. We expect, as a result, that the performance as a fraction of peak speed will be comparable to that of a vector machine. The machine is scalable by design, and is recognized to be very cost effective per teraflop of sustained performance. The machine should excel especially for problems for which large numbers of time steps are the figure of merit.

The primary perceived limitation of the QCDOC concerns its wider usability for non-QCD applications. We focus on Molecular Dynamics (MD) algorithms with long-ranged (Coulomb) potentials to address this issue. We validate our basic claim that QCDOC is an efficient platform for MD by the construction of a performance model for MD algorithms for this architecture and by its partial validation. Performance models address limits of finite computational and communication speed, and memory size. When machine design parameters are inserted into these performance models, we obtain a prediction of machine performance. Our conclusion is a surprising prediction of unusually long simulated times for MD problems with an interesting number of atoms. In view of the attractive cost, power, and cooling requirements, our conclusion raises the possibility that the QCDOC architecture is a viable model for computers for a range of problems of general interest. Validation includes not only the speed and memory of the algorithm, but also the accuracy of the calculation as a result of the various approximations (numerical cutoffs) it employs. Thus scientific validation is a component of the performance model.

MD simulations [3, 4] are ideal for massively parallel computing because the time spent on communication can be a small fraction of the time spent on computation. However, most MD simulations have been performed on machines where this potential has not been realized because of slow communication and/or a small number of processors.

For a class of MD simulations characterized by short range forces we find that calculations on 10^6 atoms with simulated times up to 10^{-5} seconds are feasible with 5 weeks of computing time. Including long range (Coulomb) interactions, we project simulation of 10^5 particles for 10^{-6} seconds within 11 days.

2. QCDOC and its Performance Characterization

2.1. Hardware Characterization

The QCDOC machine consists of 32 bit IBM PowerPC 440 processors running at 500 MHz. These processors achieve 1 Gflops peak (two double precision operations per clock cycle), using an integral 64 bit floating point unit. The crucial design feature is a set of 24 serial communication channels to neighboring nodes in a 6D torus communication network, which are capable of sending or receiving data at the rate of 500 Mb/s per channel with a latency of 95 nanosec (send) and 320 nanosec (receive), resulting in an aggregate bandwidth of 12Gb/s for each node. Each processor has 4 megabytes of embedded (on chip) memory (EDRAM) with a memory to processor bandwidth of 8 Gb/s [1].

Observe that separate units on the chip control communication and floating point arithmetic. (See Fig. 1.) For this reason, the communication time is largely shielded from the computations, and we make the simplifing assumption that the two can be run totally concurrently. This assumption is specific to the QCDOC architecture and is not necessarily valid for other machines.

We will place all data in the embedded memory, and with its high bandwidth, we expect to achieve a high fraction of peak performance. However, in order to project conservative estimates, we use performance efficiency based on main memory. We tested our Ewald MD code on a Sparc Solaris, achieving a fraction $e_{\text{efficiency}} = 0.3$ of peak power for long range (Ewald) MD calculations with10^3, 20^3, 30^3, and 48^3 particles on a single processor. Accordingly, we will use these values in our performance estimates for the proposed QCDOC.

2.2. Performance Characterization

Now, we consider the QCDOC performance characterization. The computational time $t_{\text{computation}}$ is given by

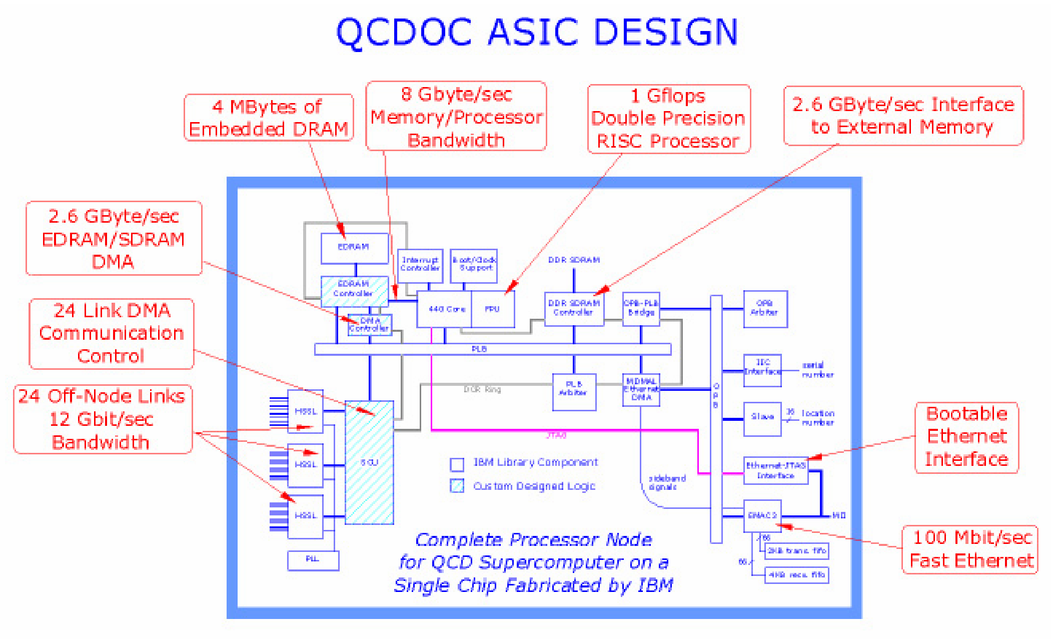
$$t_{\text{computation}} = N_{\text{ops / processor}} / (e_{\text{efficiency}} \times f) \tag{2.1}$$

where $N_{\text{ops / processor}}$ is the number of operations per processor, $e_{\text{efficiency}}$ is the single processor efficiency, expressed as a ratio of sustained to peak single processor performance, and *f* is the single processor peak performance. Here and below, op designates a floating point operation, as distinguished from the conventional abbreviation flops for floating point operations per second. The raw data for this paper has been determined by hand counts of operations (as the code is fairly simple), using the following rules: The actual total counts of basic operations such as sin(), cos(), exp(), division, multiplication, addition, and subtraction are determined. For achieving single-procession accuracy, 8 operations (ops) are required for sin(), cos(), exp(), and 15 for erfc() while 4 operations are needed for division. Multiplication, addition, and subtraction are each defined as requiring one operation.

Our hardware communication model is likewise very simple. We adopt the model

$$t_{\text{message}} = b^{-1}x + l \tag{2.1}$$

for the time for a single message sent over a single channel, where $b = 0.5 \times 10^{9}$ is the bandwidth (bits/sec) per mono-directional channel, $l = 320 \times 10^{-9} = 3.2 \times 10^{-7}$ seconds is the latency and x is the message size in bits.



Misson-critical, custom logic (hatched) for high-performance memory access and fast, low-latency off-node communications is combined with standards-based, highly integrated commercial library components.

Figure 1 The main components with connectivity of the QCDOC ASIC node.

Let us apply this formula to develop a model for spatially nearest neighbor communication. We need one message hop per coordinate direction to communicate to coordinate-wise nearest neighbors. Thus to reach the off-diagonal coordinate-wise nearest neighbors, we need 3 hops in three spatial dimensions. If $x_{\text{processor}}$ is the per processor message size, then the bandwidth applies to the total of these three messages. There are $27-1=26$ such units of data to be sent and received at each processor, distributed over the 3 messages. Thus the time t_{nn} for nearest neighbor communication is give by

(2.3) $$t_{\text{nn}} = 26 \times \frac{2}{wb} \times x_{\text{processor}} + 3l$$

where $w=12$ is the number of one way wires communicating to coordinate-wise neighbors in the 3D spatial lattice.

Memory management is also an issue. For short range forces, the algorithmic aspects of message memory management are easily handled. The N_p particles in the central cell and in each of itsl 26 nearest neighbor cells will be stored, with a total storage use of $24 \times 27 \times N_p = 64.8KB$ based on the storage of three double precision coordinates per particle. Here we have assumed $N_p = 100$. No special message buffers are needed, as the messages can address these storage locations directly. The first message, to spatial nearest neighbor cells in one spatial direction, has no redundancy, and no potential writing conflicts. By this we mean that the multiple messages received at each node are all unique. For the second message, each piece of data to be received can potentially come from two cells, in the sense that the data originating at processor A can arrive in two hops at processor B via two distinct paths. Thus there is a potential to receive the same data twice, i.e. nonuniquely. We order the coordinate directions in a systolic order. The second message will send data which differs in one of the three coordinates from the sending cell. It will send it to neighbors in the directions of the first (in cyclic order) of the two consecutive coordinates that agree with the sending cell. Thus the second message only sends half of the data potentially available for transmission through each spatially related channel to avoid a writing conflict on receipt. The third message transmits data which differs in two of its coordinates from the location where it is received, and when received, the data differs in all three coordinates. In view of the small size of the messages, we choose a suboptimal strategy and send the third message on the first coordinate direction only. This message consists of data agreeing with the sending processor in this direction only. This avoids write conflicts on arrival as each piece of data arrives uniquely. Similar but more complex memory management issues arise with the larger all to all messages of the long range MD algorithm.

The all to all communication requires further analysis. For an all to all communication, we adopt the same formula, with effective, all to all values for b^{-1} and l,

(2.4) $$t_{\text{all to all}} = b^{-1}_{\text{all to all}} x + l_{\text{all to all}} .$$

The all to all bandwidth is independent of the number of message hops. In fact $b_{\text{all to all}} = wb/2$ where $w=24$ is the number of wires per processor ($w/2$ is the number of wires per direction per processor) and b^{-1} is the single channel one directional bandwidth. Here x is the total message size at a given processor. Thus for P processors, $x = (P-1)x_{\text{processor}}$.

The all to all latency is $l_{\text{all to all}} = n_{\text{hops}} l$, where l is the single channel latency introduced above and n_{hops} is the number of hops for an all to all message. For a network in the form of a torus in dimension D with each factor periodic (a circle) of size r_i, $[r_i/2]$ hops are required to traverse a single factor of the torus and

(2.5) $$n_{\text{hops}} = \sum_{i=1}^{D} [r_i/2]$$

hops, or $D[r/2]$ hops for a homogeneous network (all $r_i \equiv r$), to traverse the full network. Here $[a]$ denotes the integer part of the real number a.

The finite memory and thus message buffer size on each processor provides an additional contribution to latency. Messages may have a very large size. As a result they will be broken up into a series of messages of the size determined by the message buffers, giving rise to an additional latency contribution. This correction can be absorbed in a renormalization of the bandwidth *b*. We enforce a maximum message size, which then modifies the bandwidth. The point is to choose this maximum large enough so that the bandwidth is effectively not changed. We now see that such a maximum message size can be chosen to be rather small. Even in the extreme case of a message of size as small as 10^4 bits (160 double precision words), the message transmission time is

$$(10^4/b) + l = 2\times10^{-5} + l = 2\times10^{-5} + 3.2\times10^{-7} = 2\times10^{-5} \text{ sec}$$

and the maximum message size latency makes a negligible contribution to the total transmission time.

Thus the time for an all to all communication is

(2.6) $$t_{\text{all to all}} = \frac{2x}{wb} + n_{\text{hops}} l$$

The success of the QCDOC architecture can be seen from this formula, as the number

$$n_{\text{hops}} = 3\sqrt[6]{P}$$

of hops is small, for P processors arranged in a 6*D* torus. The total latency $n_{\text{hops}} l$ is small. The communication time is scalable and is dominated by the bandwidth. Here the parallelism of the $w = 24$ fast communication channels per processor becomes important.

3. Molecular Dynamics

In classical MD the positions of a set of interacting particles are advanced according to Newton's second law

(3.1) $$M_i \frac{d^2\vec{r}_i}{dt^2} = \vec{F}_i$$

where r_i is the position of the *i*th particle, M_i its mass and $\vec{F}_i$ the force acting on it due to the other particles. Often the 2nd order difference equation

(3.2) $$\vec{r}_i(t+\Delta t) = 2\vec{r}_i(t) - \vec{r}_i(t-\Delta t) + \vec{F}_i / M_i \Delta t^2$$

is used, where Δt is the time step, usually of the order of 10^{-15} seconds. Simulations are typically run for 10^4 - 10^5 time steps producing 10^{-11}-10^{-10} seconds of simulated time. This is less than a nanosecond, and it is a major limitation since many processes of interest occur on time scales of 10^{-6}-10^{-3} seconds or longer.

Another issue concerns the size of the sample. While a great deal has been learned about the thermal properties of materials from simulation cells containing several thousand atoms, current interest extends to 10^6 or more [6]. Typical biological molecules contain 10^5 atoms [7]. A metal or oxide particle 10 nanometers in diameter contains approximately 50,000 atoms. Simulations with many nano particles and up to 10^6 atoms have been performed [8].

In order to evaluate Eq. (3.1) or (3.2), we need to calculate the force on each particle. The force computation is the most time consuming portion of the entire computation. The force is usually obtained from a density dependent or embedded atom force law [8]. These force laws are semi empirical, which means that they must be calibrated against experimental data or parameters calculated for example from density functional theory. Local force laws are usually cut off smoothly so that they are zero beyond 1 or 2 particle neighbor distances. However, Coulomb forces are global, and couple all the atoms in the simulation.

The basic strategy for parallelizing the MD code is to break the sample into subdomains with a subset of particles assigned to each processor. To evaluate the forces it is necessary to communicate the positions of the particles in other cells with which a given particle interacts. Similar parallelization schemes have been applied previously [10, 11, 12], but not on machines with the capability of QCDOC.

The highly optimized parallel code NAMD [13] loses parallel scaling at about 200 particles per processor, whereas we analyze a case with 14 particles per processor and have not yet reached the parallel scaling limit. The fine scale parallelism is important not only to allow effective use of more processors. With fine scale parallelism, all data is located within L2 chache, so that high levels of single processor efficiency can be achieved.

4. A Performance Model for MD with Short Range Forces

We next develop a performance model for an MD algorithm involving short-ranged forces. Combining this with the hardware performance model of Sec. 2 gives us an application performance model, allowing the prediction of simulation times achievable running MD codes on the QCDOC. We consider two cases. The first, considered in this section, has short range forces only. Typical nanoscience materials problems are of this nature, and for this case, the choice of $N = 10^6$ atoms defines a typical to large sized problem. The second case, with long range forces, is targeted at typical biological problem sizes (with 10^5 atoms). In both cases, we aim to maximize the length of time simulated.

Maximizing the simulated time requires that the minimum number N_p of particles be allocated to each processor. We choose $N_p \sim 100$. Fewer than this number may require communication beyond nearest neighbor processors. We consider an MD cell containing $N = 10^6$ particles distributed over $P = 10,000$ processors. We define $N_{\text{ops / particle}}$ as the number of operations per particle required to evaluate the force for a short range potential. $N_{\text{ops / particle}}$ is a function of the particle density (which determines the number of particles within the range of the short range forces) and the allowed error (which determines the distance at which a rapidly decaying force of infinite extent can be cutoff).

As a simple illustration, we summarize this dependency in Table 1, showing $N_{\text{ops / particle}}$ *vs.* *E*, the relative RMS solution error in the force per particle. Here we use a simple Lennard-Jones (LJ) 6-12 potential. We set both coefficients for the LJ potential to be unity. To fix the density units, we consider particles in a random perturbation from a regular lattice which has unit spacing between the particles. The error in the force calculation results from truncating the energy at a distance $R_{\max}$. Because we consider particles at locations perturbed from a unit lattice, the distance $R_{\max}$ is expressend in units of mean interparticle spacings,, and is thus a count of the number of neighbors included per coordinate direction within the cutoff. (The same density convention and choice of units for $R_{\max}$ will be used in Sec. 5.) With this convention, the minimum of the LJ potential occurs at distance 1, and the standard LJ parameter $\sigma = 1$. The error is expressed dimensionlessly as a ratio of RMS error per particle, divided by the true force. The LJ error results from truncation of the r^{-6} term. For this reason the errors have a common sign and there is no cancellation of errors. The error equals the sum of the omitted terms. Results for a test based on 1000 particles are summarized in Table 1. Obviously, more complex potentials, such as the molecular biology potentials of CHARMM, would have a different operation count. Since the LJ potential has the longest range of commonly used short range potentials and since it is unusual in having a fixed sign of error, we expect the other contributions to the CHARMM short range potentials to have smaller operation counts.

$R_{\max}$	$N_{\text{ops / particle}}$	RMS Relative Error
3	509	.001727
4	1200	.000227
5	2350	.000031
6	4071	.000005

Table 1 Operation count statistics for short-ranged LJ 6-12 potential.

Based on a arbitrary but probably reasonable choice of accuracy (RMS relative error), we refer to Table 1 to determine the number of ops/particle. On this basis we assume $N_{\text{ops / particle}} \sim 10^3$ operations / particle, giving $N_{\text{ops / particle}} \times N_p = 10^5$ ops / processor. Assuming processor efficiency $e_{\text{efficiency}} = 0.3$ and single processor peak speed $f = 10^9$ flops, the update time is

(4.1) $$t_{\text{computation}} = N_{\text{ops / particle}} \times N_p / (e_{\text{efficiency}} \times f) = 10^5 / (0.3 \times 10^9) = 3.33 \times 10^{-4} \text{ sec / step .}$$

We next estimate communication times. We assume it would be necessary to communicate (send and receive) the positions of all the atoms in the cell to its neighbors, for a local force law. Based on the proposed problem size and number of processors, we expect 4 to 5 particles per linear dimension per processor. Thus from Table 1, we see that (depending on the assumed level of accuracy required), communication of all data from all nearest neighbor processors will be required to give sufficient data to a given processor. Only some of this data will be used for any specific atom, but basically all will be used for the collection of all atoms on the processor. For local communications, we use only the 3*D* spatial sub-lattice of the full $D = 6$ dimensional communication lattice. Thus we set $w = 12$. QCDOC can simultaneously send and receive data to/from each of its 6 spatial neighbors in three spatial dimensions at a rate $b = 0.5 \times 10^9$ bits/sec. To reach the $27 - 1 = 26$ neighbors in the three spatial coordinate directions requires three separate messages. The first message has a size

$$x_{\text{processor}} = x_{\text{sin gle particle}} \times N_p = 3 \times 64 \times N_p = 192 \times N_p \text{ bits}$$

in view of the three double precision numbers of size $x_{\text{single particle}} = 3 \times 64 = 192$ bits used to describe each particle. The time for a local communication step to the 26 spatial neighbor cells is thus

(4.2) $$\begin{aligned} t_{\text{nn}} &= 26 \times (1/6b) x_{\text{procesor}} + 3l \\ &= 26 \times (1/6b) \times 192 \times N_p + 3l \\ &= 832 \times (N_p / b) + 3l = 1.66 \times 10^{-4} + 9.6 \times 10^{-7} = 1.67 \times 10^{-4} \text{sec / step .} \end{aligned}$$

The communication and the computation times are comparable. The total time for computation and communication for a short range time step is

(4.3) $$t_{\text{short range}} = \max\{t_{\text{computation}}, t_{\text{nn}}\} = 3.33 \times 10^{-4} \text{ sec / step}$$

because computation and communication can run concurrently. (See the discussion in Sec. 21.)

Assuming time integration steps of 10^{-15} seconds (time for the simulated system), 10^{-5} seconds of simulated time requires 10^{10} steps, or approximately

(4.4) $$10^{10} \text{ steps} \times 3.33 \times 10^{-4} \text{ sec / step} \times \left(8.6 \times 10^4 \text{ sec / day}\right)^{-1} \text{ days} = 39 \text{ days .}$$

Because the latency is negligible in this analysis, 10^{-3} seconds of physical time would be achieved on a hypothetical machine with 10^6 processors.

5. A Performance Model for MD with Coulomb Potentials

Simulations involving long-ranged interactions, usually characterized by the Coulomb potential, have been regarded as a serious challenge to QCDOC due to the inherent nearest neighbor connectivity of its network topology. We demonstrate here this machine's usability to MD involving long-ranged interactions by deriving a performance model for the conventional Ewald algorithm and particle-mesh Ewald algorithms.

5.1 The Ewald Algorithm

We develop a performance model for the Ewald algorithm, which we use to handle long range forces. The algorithm scales as $O(N^{3/2})$ rather than the less favorable scaling$O(N^2)$ of the direct solution method. The multipole algorithm has a still more favorable scaling, $O(N \ln N)$, but studies [14] have shown no advantage in the use of this more complex algorithm for problem sizes typical of biological systems, and certainly not for the problem sizes we consider. A refinement [15] to the Ewald formula has been proposed to exclude all of the short range forces from the *k*-space sum, as the short range forces are rapidly varying in time. When they are excluded from the *k*-space sum, this more expensive operation can be performed with a larger time step, chosen here to be a multiple 10 of the basic time step. Particle mesh Ewald algorithms, also with $O(N \ln N)$ scaling, are popular. We considered those algorithms in Sec.6.

The Ewald algorithm, having few numerical parameters, allows a straightforward estimate of timing. The Ewald algorithm expresses the Coulomb interaction energy *V* due to charged particles in a cube of side *L* with periodic boundary conditions as

$$V = (1/2)\sum_{i,j}\left(V_{ij}^{r} + V_{ij}^{k}\right) \tag{5.1}$$

where

$$V_{ij}^{r} = q_i q_j (1-\delta_{ij}) \frac{\operatorname{erfc}(\alpha r_{ij})}{r_{ij}} \tag{5.2}$$

$$V_{ij}^{k} = q_i q_j \frac{1}{\pi L^3} \sum_{k\neq 0} \frac{4\pi^2}{k^2} \exp\left(-\frac{k^2}{4\alpha^2}\right) \cos(k \cdot r_{ij}) - \delta_{ij} q_i q_j \frac{2\alpha}{\sqrt{\pi}} \ , \tag{5.3}$$

following the presentation of [15], where the non-obvious aspects of the terminology in (5.2)-(5.3) are explained. The time step optimization [15] results from the use of a smooth cutoff function *h* to isolate the local and rapidly varying part of the short range forces, through use of the formulas

$$
\begin{aligned}
\frac{1}{r} &= \frac{1}{r}[\mathrm{erf}(r) + \mathrm{erfc}(r)] \\
&= \frac{1}{r}[h + \mathrm{erf}(r) + \mathrm{erfc}(r) - h] \qquad (5.4) \\
&\approx \underbrace{\frac{h}{r}}_{\text{short } \Delta t} + \underbrace{\frac{1}{r}[\mathrm{erf}(r) + (\mathrm{erfc}(r) - h)]}_{\text{Long } \Delta t}
\end{aligned}
$$

Here the first term, h/r, is the local and rapidly varying part of the potential. It is integrated with a fast time step to achieve numerical accuracy. The second term has these same terms explicitly removed. It is the long range, slowly varying part of the potential. It is integrated with a slow time step, assumed here to be a multiple 10 of the fast time step. This division of terms is helpful, as the second term involves more operations, and is the dominant part of the computation. The range of the local cutoff function *h* and the ratio of fast to slow time steps are, of course related quantities. The term erf(*r*) is analyzed in Fourier space, and is referred to as the *k*-space term. The term erfc() is referred to as the real space term. The term h/r is a multiple of the local (fast time step) term, and does not need to be recomputed.

The basic steps in the algorithm are:

1. Compute the local, real space part of the forces with data coming within the range of the cut off function h. For fast time steps, the potential has only the h/r component while for slow time steps it has a $(1/r)[\mathrm{erfc}(r) - h]$ term, the latter to be combined with the *k* space terms.
2. Communicate the particle positions r_j and charges q_j all to all.
3. Assign *k* values to processors, or rather (since there are fewer *k* values than processors), processors to *k* values. Separate the r_i and r_j dependencies in $\cos(k \cdot r_{ij}) = \cos(k \cdot r_i - k \cdot r_j)$ through use of a double angle formula. For each *k* value, sum over positions r_j.
4. Communicate all the resulting *k* value terms to all processors needing an r_i force value. This communication is also all to all. The previous communication can be regarded as going from all r_j to all *k*, and this goes from all *k* to all r_i.
5. Sum over all *k* values for each position r_i.
6. Update velocities and positions.

Only steps 1 and 6 are performed for fast time steps, while all are performed for slow ones. Assuming $O(N^{1/2})$ *k* values, the steps 3 and 5 are $O(N^{3/2})$, and these are the limiting steps of the algorithm. Thus we will show that the algorithm is floating point limited, and is not communication (latency or bandwidth) limited for the problem and machine size we propose. Since we will find fewer k values than particles in a balanced algorithm, the time limiting communication step is step 2. Likewise the communication time for step 1 is smaller still, even though there are more of these steps. Due to limits of local memory, steps 2 and 3 are interleaved, as are steps 4 and 5.

5.2 Communication Time Estimates

We now set $N_p = 14$ particles / processor. Assume $P = 4^3 \times 5^3 = 8 \times 10^3$ processors arranged in a six dimensional torus, with 3 factors of size $r_i = 4$, and 3 of size 5, and $N = N_p \times P = 14 \times 8 \times 10^3 = 1.12 \times 10^5 \approx 10^5$ particles. Using (2.5) we compute $n_{\text{hops}} = 3 \cdot [r_1 / 2] + 3[r_2 / 2] = 3 \cdot 2 + 3 \cdot 2 = 12$. This level of parallelism is highly aggressive. The success of this choice will be seen from the fact that the communication time is dominated by the computational time.

The communication time for an all to all communication is the time to get the data onto (or off of) a single processor. But the communication (bandwidth) time is not dependent on the number of hops. Each piece of data is needed and used at each of the intermediate processors, following a so-called store-and-forward protocol, as it moves to its destination. The communication time, charged only when the data is used, is always a single hop communication time. Thus the all-to-all communication time is

$$\begin{aligned} t_{\text{all to all}} &= (2/wb) \times x_{\text{single particle}} \times N + n_{\text{hops}} l = (1/6) \times 10^{-9} \times (3.5 \times 64 \times N) + 12 \times 320 \times 10^{-9} \\ &= 3.7 N \times 10^{-8} + 3.84 \times 10^{-6} = 4.2 \times 10^{-3} + 3.8 \times 10^{-6} \approx 4.2 \times 10^{-3} \text{ sec / step} . \end{aligned} \tag{5.5}$$

Here we have included one integer to label the particle charge, in addition to the three doubles which describe the particle positions, or a total of 3.5 doubles per particle storage. There are two all to all communications in each Ewald time step. One is from all *x* to all *k,* and is estimated as above. The other, from all *k* to all *x* is proportional to the number of *k* values, estimated as $\sqrt{N}$ below, and is thus negligible. Thus the time $t_{\text{all to all}}$ estimated above is also the total communication time for a Ewald time step.

5.3 Truncation Error Analysis

Next, we estimate the number of terms and the time spent in evaluation of the Ewald sums. The number of these terms determines the time spent in the two computational steps. These are the rate limiting steps of the algorithm.

For the real space part of the sum, we assume that the cutoff function *h* extends to a distance of 3 particle neighbors. Then there are a total of $7^3 / 2 = 172$ short range particle pair interactions to be computed per particle. They are communicated from $5^3 = 125$ processors, but as observed elsewhere, communication issues are not the limiting factor. This number of terms has to be computed with a fast time step. Since this number of terms is smaller by a factor of about 10 relative to the slow time step terms, we thereby compensate for the 10 fold increase of work due to the fast time step.

The slow time step terms have two pieces. A real space sum with the erfc() summand and a *k* space sum with summand exp() ×cos(). We estimate the number of terms present in each. These terms contain all *r* and *k* terms up to limits $r_{\max}$ and $k_{\max}$ respectively. These limits are determined by the required accuracy. The limits are a function of that accuracy and the Ewald splitting parameter α. With the accuracy fixed, we adjust α so that each sum represents an equal number of ops. In the leading order asymptotic solution of the equations, the resulting α is independent of the error E. First we fix $r_{\max}$ and $k_{\max}$. Recall that, as in Sec. 4, these numbers count the number of terms per coordinate direction within the range of the cutoff.

Let E^r denote the real space contribution to the RMS average error in the force per particle. We choose integer charges q_i randomly from $\{\pm 1, \pm 2, \pm 3\}$. Then the RMS average charge is $\| q \| \equiv \langle q \cdot q \rangle^{1/2} = \sqrt{14/3}$. Thus

(5.6)
$$E^r = \frac{1}{2} \| q \|^2 | \sum_{|r_i| > r_{\max}} \text{Grad erfc}(-\alpha r_i) / r_i \, |$$

Using the average density of points N / L^3 and the asymptotic expression

(5.7)
$$\text{erfc}(x) \sim \frac{\exp(-x^2)}{x} \quad ,$$

we have

$$E^r = \frac{7}{6} \frac{4\pi N}{\alpha L^3} \int_{r > r_{\max}} 2 \exp(-\alpha^2 r^2) \left[\alpha r + 1/\alpha r \right] dr$$
$$= \frac{14\pi N}{3(\alpha L)^3} \exp(-\alpha^2 r_{\max}{}^2) \left[\alpha + 1/\alpha r_{\max}{}^2 \right]$$

where we have simplified the analysis by evaluating the *r* and1/*r* factors at the lower bound before performing the integral. Then

$$-\ln E^r = \alpha^2 r_{\max}^2 + \left[\ln (\alpha^3 L^3 / \pi N) + \ln (3/14) \right] - \ln [\alpha + 1/\alpha r_{\max}{}^2]$$

For simplicity, we analyze this formula using only the leading order asymptote,

(5.8)
$$-\ln E^r = \alpha^2 r_{\max}^2 \quad .$$

We choose $r_{\max} = (1/\alpha)(-\ln E^r)^{1/2}$.

We next estimate the per particle RMS error from termination of the *k*-space Ewald sum at $k_{\max}$. We use double angle formulas to replace the factor $\cos(k \cdot r_{ij}) = \cos(k \cdot (r_i - r_j))$ by cos and sin terms in the arguments $k \cdot r_i$ and $k \cdot r_j$. The first of these factors is out of the sum over *j* and *k*. The sum of the second is multiplied by the charge q_j. The sum over *j* defines the Fourier (sine or cosine) transform $\hat{q}(k)$. This gives an expression of the form

(5.9)

$$E^k = \frac{1}{2} \| q \| \times | \sum_{|k|>k_{\max}} \widehat{q(k)} \frac{4\pi^2}{k^3} \exp(-k^2/4\alpha^2) |$$

$$\approx \frac{1}{2} \| q \| \times | \int_{|k|>k_{\max}} \widehat{q(k)} \exp(-k^2/4\alpha^2) \left(\frac{L}{\pi}\right)^3 \frac{4\pi^2}{k^3} d^3k |$$

where we make use of the density $(L/\pi)^3$ of points in *k* space. We know that q_j is bounded as a function of *j*, and thus $q_j/(1+j^2)$ is in L_2, with a norm proportional to $\| q \|$. In Fourier space, we integrate by parts twice, and then apply the Schwartz inequality to separate the $\hat{q}$ and exp factors. Taking only the leading order term in the asymptotic analysis of the resulting expression, we have

(5.10) $$-\ln E^k = k_{\max}{}^2/4\alpha^2$$

We actually want to analyze the relative and not the absolute error. However, division by the mean per particle force contributes a lower order term, and does not appear in the leading order asymptotic error.

Note that α and $E = E^r = E^k$ are the only parameters in the expansion.

5.4 Truncation Error: Numerical Experiments

We have performed several numerical experiments to validate the performance models. In our experiments, we distribute $N = 48^3 = 110,592 \approx 10^5$ particles in a 3D box of size $L^3 = 48 \times 48 \times 48$. Their coordinates are random perturbations, by 10%, of the inter-particle unit distance or mesh size, from cubic lattice points. The mass of each particle is normalized to 1. Their charges are random integers selected from $\{\pm 1, \pm 2, \pm 3\}$. In measuring the loads, we count the number of floating point operations, instead of absolute time in seconds, to avoid machine-dependence for performance models.

Tables 2 and 3 summarize the results of our numerical experiments. In these tables, we give the RMS relative truncation errors, for comparison to formulas (5.8) and (5.10). We show relative RMS error in the *r*-space and *k*-space contributions to the force field, and the number of operations f^r and f^k needed for the evaluation of the truncated approximations to these force fields. In order to compute the errors E^r and E^k, we need an exact value for the real and *k*-space terms. This is obtained by summing over all indices in (5.1) and (5.2). In each set of the experiments, we select five equally-spaced values for $\alpha \in [0.1, 0.3]$. For each value of *α*, we select six values for $r_{\max} \in [10, 20]$ for estimating E^r and six values for $n_{\max} = k_{\max}(L/2\pi) \in [3,8]$ for estimating E^k. These choices are made in order to find the optimal parameters which minimize the total floating point operations $f = f^r + f^k$ per particle per step at given error.

In the following two tables and the three figures, the numbers are expressed in the usual scientific notations. In other words, $1.234E-3=1.234\times10^{-3}$. Also, in the entire article, we have used both $\ln(x)$ and $\log(x)$ which mean the e-based and 10-based logarithm of x respectively.

$r_{\max}$	α =0.1	α =0.15	α =0.2	α =0.25	α =0.3	f^r
10	1.017E-1	2.108E-2	2.841E-3	2.439E-4	1.321E-5	65,346
12	5.497E-2	6.198E-3	3.764E-4	1.187E-5	1.902E-7	106,847
14	2.800E-2	1.592E-3	3.758E-5	3.583E-7	1.350E-9	161,554
16	1.251E-2	3.333E-4	2.744E-6	6.741E-9	4.841E-12	228,433
18	5.281E-3	5.895E-5	1.470E-7	7.850E-11	8.848E-15	307,635
20	2.092E-3	8.911E-6	5.817E-9	5.577E-13	3.453E-17	398,505

Table 2 The *r*-space error E^r dependence on α and $r_{\max}$ and the floating-point operation counts f^r varying with $r_{\max}$.

$n_{\max}$	α =0.1	α =0.15	α =0.2	α =0.25	α =0.3	f^k
3	4.440E-3	4.266E-2	1.012E-1	1.564E-1	2.020E-1	5,655
4	2.003E-4	9.702E-3	3.956E-2	7.846E-2	1.164E-1	13,404
5	2.840E-6	1.199E-3	1.064E-2	3.054E-2	5.579E-2	26,180
6	1.530E-8	9.827E-5	2.315E-3	1.057E-2	2.501E-2	45,239
7	6.039E-11	7.892E-6	5.270E-4	3.882E-3	1.191E-2	71,838
8	7.509E-14	3.599E-7	8.537E-5	1.137E-3	4.825E-3	107,233

Table 3 The *k*-space error E^k dependence on α and $k_{\max} = 2\pi n_{\max} / L$ and the floating-point operation counts f^k varying with $n_{\max}$.

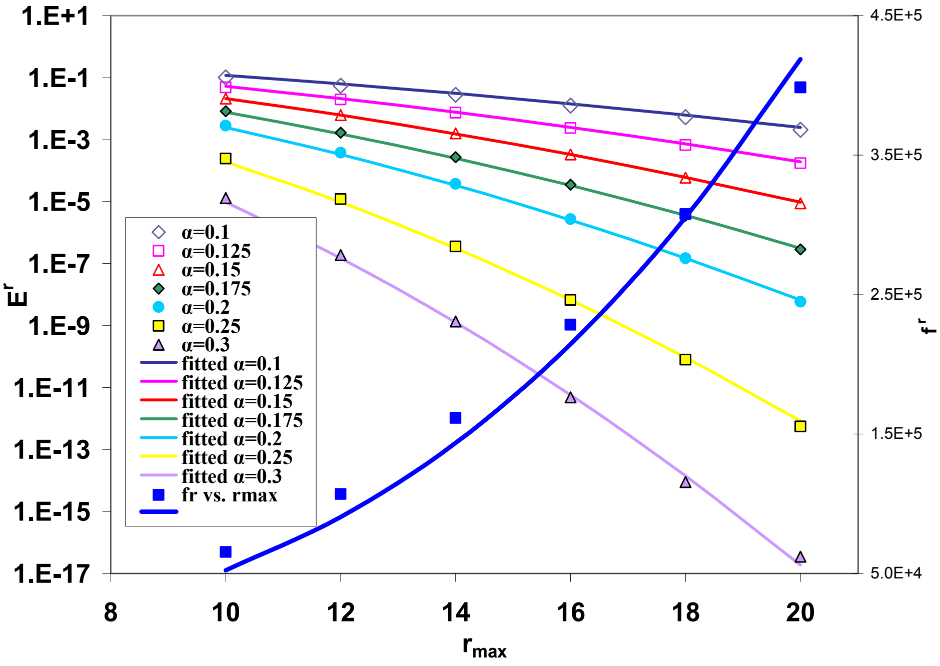


Figure 2 Graphical presentation of data in Table 2 (r-space) and formulas fit to these data. The curves are the best fits to the experimental results shown as points. The left vertical axis is for the error E^r , the right is for the operation counts f^r and the horizontal axis is for $r_{\max}$.

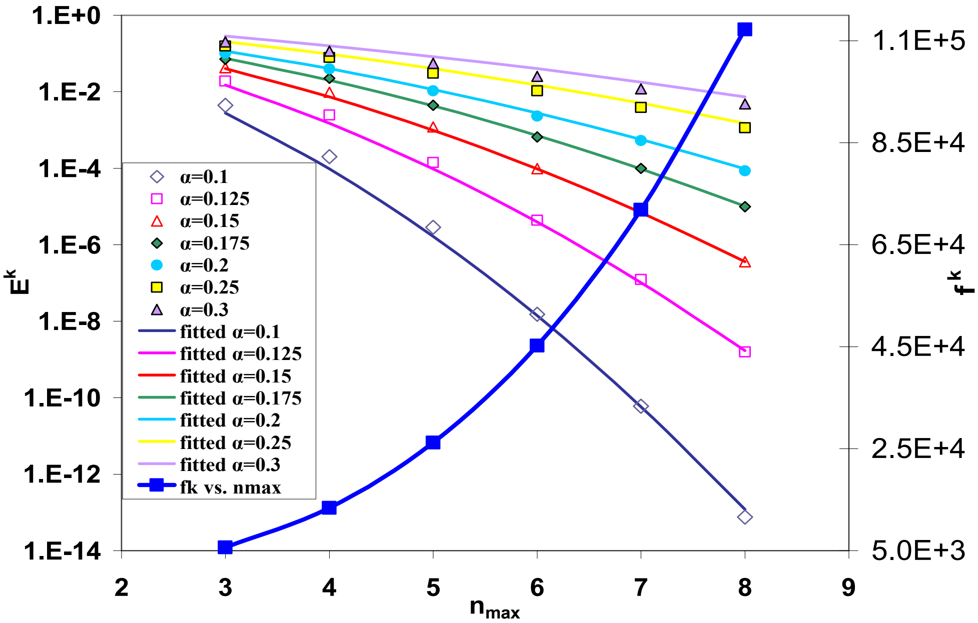


Figure 3 Graphical presentation of data in Table 3 (k-space) and formulas fit to these data. The curves are the best fits to the experimental results shown as points. The left vertical axis is for the error E^k , the right is for the operation counts f^k and the horizontal axis is for $n_{\max}$.

Using the results in Tables 2 and 3, we fit E^r to a function of two variables (depending on α and $r_{\max}$), and we fit E^k to another function with two variables (depending on α and $n_{\max}$). It is not difficult to justify the choice of a bi-quadratic form as the fitting function. For r-space and k-space, we tried

$$-\ln(E^r) = (c_1 + c_2\alpha + c_3\alpha^2)(d_2 r_{\max} + d_3 r_{\max}^2)$$

$$-\ln(E^k) = (c_1 + c_2\alpha^{-1} + c_3\alpha^{-2})(d_2 n_{\max} + d_3 n_{\max}^2)$$

It turns out that only two terms, ($r_{\max}$α) and $(r_{\max}\alpha)^2$ for r-space and ($n_{\max}$/α) and $(n_{\max}/\alpha)^2$ for k-space, produce significant coefficients while other coefficients simply vanish. Similarly, we fit the operation counts f^r and f^k to $r_{\max}$ and $n_{\max}$, respectively, in polynomials up to the 3rd order. In all fitting cases, the relative χ^2 errors are less than 1.5% except the operation count f^r for which we over-fit by nearly 10%.

The r-space error E^r is well fit by the expression

(5.11) $$-\ln(E^r) = 0.855(r_{\max}\alpha)^2 + 1.29(r_{\max}\alpha)$$

and its associated operation count f^r is fit by

(5.12) $$f^r = 52 r_{\max}^3$$

Similarly, the k-space error E^k is well fit by:

(5.13) $$-\ln(E^k) = 0.003514\frac{n_{\max}^2}{\alpha^2} + 0.09037\frac{n_{\max}}{\alpha} = 0.820\frac{k_{\max}^2}{4\alpha^2} + 1.38\frac{k_{\max}}{2\alpha}$$

and its associate operation counts f^k is fit well by

(5.14) $$f^k = 209 n_{\max}^3$$

The discrepancies between the theory and experiment errors, measured by $\max|E^r_{Theory} - E^r_{Exp}|/E^r_{Theory}$, are 70%. These discrepancies could be reduced through inclusion of the lower order terms in the theory, a step not pursued in this work.

The above fitted formulas can be used to provide guidance for correct choices of Ewald parameters and to predict the operations required for a given set of parameters. These calculations can also be carried by our theoretical formulas. But, in using theoretical formulas for such estimates, we must increase the expected errors by 70% to compensate the discrepancy.

5.5 Operation Count Analysis

In this section, we derive a formula for total operation counts as a function of overall error and the Ewald splitting parameter α. We set $-\ln(E^r) = -\ln(E^k) = \varepsilon$. We eliminate $r_{\max}$ from (5.11) and (5.12) to express f^r in terms of ε and α. Similarly, we eliminate $n_{\max}$ from (5.13) and (5.14) to express f^k in terms of ε and α. The resulting total operation count $f = f^r + f^k$ per particle per step can then be written as a function of ε and α:

$$f = \left(4\frac{\sqrt{0.49+\varepsilon}-0.70}{\alpha}\right)^3 + \left[100\alpha\left(\sqrt{0.58+\varepsilon}-0.76\right)\right]^3$$

It's easy to show that, for fixed ε, the total operation count function f is minimized at

(5.15) $$\alpha = \frac{1}{5}\sqrt{\frac{\sqrt{0.49+\varepsilon}-0.70}{\sqrt{0.58+\varepsilon}-0.76}}$$

See Figure 4.

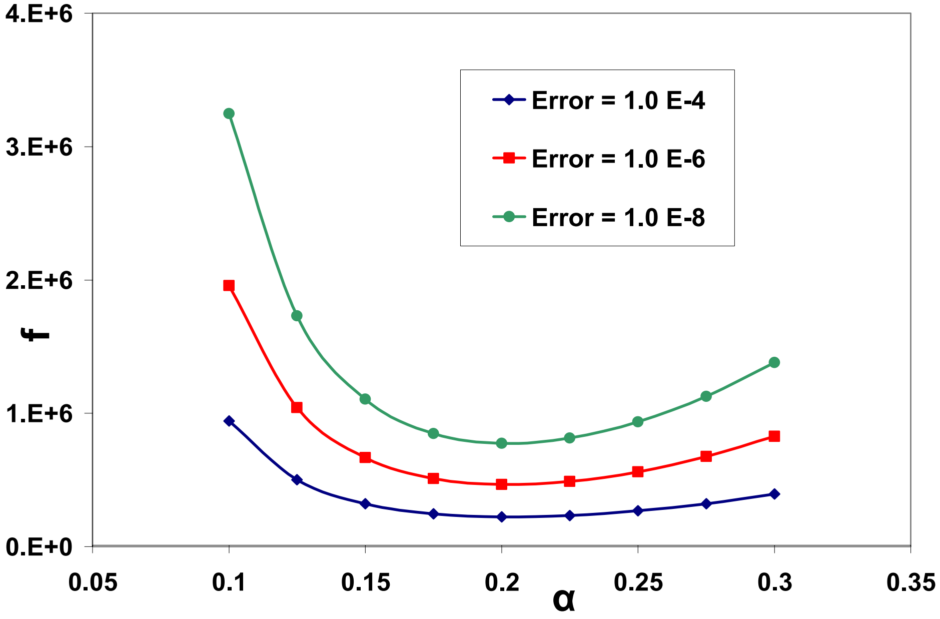


Figure 4 Total operation count as a function of α for $E = 10^{-4}, 10^{-6}, 10^{-8}$.

Using (5.15), we can find the optimal α at which $f = f^r + f^k$ is minimized for an *a priori* error $E = e^{-\varepsilon}$. We input the α and ε to the fitted error formulas (5.11) and (5.13) to find $r_{\max}$ and $n_{\max}$, resulting in a complete determination of input parameters α, $r_{\max}$, and $n_{\max}$ for optimal Ewald calculations.

Let us demonstrate the above by one concrete example. We set units so that the inter-particle distance is unity and we set $E = 10^{-4}$ thus $\varepsilon = -\ln E = 4 \times \ln(10) = 9.21$. Plugging in this value of ε to the above operation count equation, we find

$$\alpha = 0.203 \text{ and } f_{\text{opt}} = 222{,}300$$

where f_{opt} is the optimal number of floating-point operations per particle per time step. In addition, plugging in the values of α and ε to the fitted error formulas, we find

$$r_{\max} = 13 \text{ and } n_{\max} = 8.$$

The time to evaluate both the *r*- and the *k*- space terms is double the time to evaluate the *k* space terms alone, by the above choice of α. With $P = 8000$ the time for a single (long) Ewald time step is

$$\begin{aligned} t_{\text{Ewald}} &= N \times f_{\text{opt}} \times \frac{1}{P \times e_{\text{efficiency}} \times f} \\ &= 10^5 \times 222300 / [8000 \times 0.3 \times 10^9] \\ &= 9.26 \times 10^{-3} \text{sec/slow step.} \end{aligned} \tag{5.16}$$

The local terms computed on a fast time step will be analyzed in detail in a following paper, with inclusion of bonded terms and the full force terms needed for biological modeling. Here we give a very simple argument just to estimate that the local terms are not limiting. It is of course general experience that the long range forces are the limiting contributions to the force calculation. The principle of local time stepping is well established. Our proposal to use a multiple time stepping ratio of 10 is within the range favored by others [15]. The allowed ratio of slow to fast time intervals is governed by the cutoff function *h* in (5.4) and for a balanced calculation, *h* should be chosen so that the time spent on the long and short steps are comparable. Here we assume that a smooth cutoff *h* with a particle radius of 5 particles will be sufficient to allow a ratio of 10 for the slow to fast time intervals. Then we refer to the LJ estimate of Sec. 4. Since we have a factor 10 fewer particles now, the timing is reduced by a factor of 10 per fast time step. But we have 10 fast time steps per slow step, so the time estimate is actually unchanged from Sec. 4. Note that the short range LJ is more expensive than the short range Coulomb to compute, but in fact the true short range forces will include Coulomb, LJ and various bonded forces.

Thus the total time per step is estimated as

$$\begin{aligned} t_{\text{total}} &= \max \left\{ t_{\text{Ewald}}, t_{\text{all to all}} \right\} + t_{\text{short range}} = \\ &= 9.26 \times 10^{-3} + 3.33 \times 10^{-4} = 9.59 \times 10^{-3} \text{sec/step.} \end{aligned} \tag{5.17}$$

We estimate the total computation time for one micro sec of real time using 1 fsec short and 10 fsec long time steps as

$$\begin{aligned} & 10^{14} \times 10^{-6} \text{ steps } \times 9.59 \times 10^{-3} \text{ sec / step} \\ & = 9.59 \times 10^5 \text{sec} \times \left(8.64 \times 10^4 \text{ sec/day} \right)^{-1} \\ & \approx \text{ 11 days.} \end{aligned} \tag{5.18}$$

6. Particle Mesh Ewald

We discuss briefly the particle mesh Ewald method [15, 16, 17]. A particle mesh Ewald algorithm evaluates the *k* space Ewald sum by interpolating the Coulomb charges to a mesh, transforming by an FFT, solving the Poisson equation as a multiplication operator and applying an inverse FFT to evaluate the forces on an *x* space mesh. The forces are then re-interpolated from the mesh to the particle positions. This algorithm scales as $O(N \ln N)$ in contrast to the $O(N^{3/2})$ for the Ewald method. The algorithm comes in several flavors, of which the particle-particle-particle mesh Ewald (P^{3}E) method is recommended [18]. There are several parameters in this method, whose setting determines the balance between speed and accuracy. The setting of these parameters has been studied systematically [18, 19]. Accuracy can be assessed in a number of ways, a common method being RMS error in the force field. Local charge neutrality does not affect the errors, and locally they are proportional to the number of particles, *i.e.* the density. Globally, the forces decay as $O(1/r^2)$ and the error behaves as $O\left(\sqrt{\int 1/r^4}\right)$, which is convergent at infinity. Thus large systems at fixed density have a uniform error requirement. The magnitude of the allowed error is determined by comparison to the random forces (*e.g.* Brownian motion, experimentally, or a numerical heat bath, in a simulation) of the problem, among other factors. Generally, the particle mesh Ewald algorithms offer increased speed with reduced accuracy in comparison to the Ewald method, especially for large systems.

In P^{3}E, the *x* space terms are considered with a fixed cutoff, so they scale as $O(N)$. This setting would force the *k* space terms to scale as $O(N^2)$, but these are evaluated by FFTs, with $O(N_{\text{FFT}}\ \ln N_{\text{FFT}})$ scaling. The mapping of charges from particle positions to the mesh and the mapping of forces from the mesh to particle positions is local, *i.e.* $O(N)$. The FFT mesh is fixed by accuracy considerations, and at fixed density, it has an *x* space density independent of *N* for large systems. Thus $N_{\text{FFT}} = O(N)$ gives the P^{3}E method its $O(N \ln N)$ scaling. Values of $N_{\text{FFT}} = 32^3$ were used for *N* = 100 particles and a fifth order charge assignment algorithm to achieve 5 digit accuracy [18, 19]. Scaling this FFT with the mesh size *N* leads to impractical and probably unnecessarily large values of N_{FFT} so we leave N_{FFT} as a free parameter.

We assume that P^{3}E is communication dominated. In fact if this is not the case, then the P^{3}E algorithm will be a clear success on QCDOC. Thus we examine the communication pattern for P^{3}E. The method has two FFTs, one of which maps scalar charges and the other mapping vector forces. From the point of view of latency, this leads to two sets of messages. The FFTs are actually a product of 3 one dimensional FFTs, each of size $n_{\text{FFT}} \equiv \sqrt[3]{N_{\text{FFT}}}$. Each of these FFTs employs a reverse binary sort operation, to map data associated with each index in one coordinate direction to the index with the reversed binary expansion. The messages to accomplish this sort may be sent in parallel for processors with distinct values of the other coordinates. Thus we use only 4 of the 24 communication channels and count messages for a purely one dimensional problem. There are (fewer than) n_{FFT} such messages. Each has at most 2 hops. Thus we compute the FFT latency per time step as

$$l_{\text{FFT}} = n_{\text{FFT}} \times 2 \times 6 \times l$$

If $n_{\text{FFT}} = 128$, then $l_{\text{FFT}} = 4 \times 10^{-3}$ seconds, a very promising number. The bandwidth can be bounded above in terms of that for the all to all Ewald communication, and it is likewise promising, depending on the value of n_{FFT}.

A more precise assessment of the P^{3}E algorithm depends on accuracy requirements to determine n_{FFT}. We conclude that the particle mesh Ewald algorithm has the potential to allow a significant improvement over the Ewald algorithm for MD on the QCDOC. Moreover, P^{3}E will allow nearly linear scaling to larger systems of particles.

7. Conclusions

The QCDOC architecture has two attractive and unusual features: a very high degree of communication parallelism and a very low latency. The result appears to be a design with highly scalable parallel performance.

We have developed hardware and application code performance models to assess application performance on this machine. In these models, parallel scaling is maintained beyond $P = 8\times10^3$ processors running an MD simulation on 10^5 particles with long range (Coulomb) forces, computed by the Ewald method. We project over one order of magnitude in simulation speed and achievable simulation times over that obtained with other hardware. See [13] for typical benchmark resultsand the more recent refernce [20, 21].

We have two basic conclusions to draw.

1. The QCDOC is an efficient platform for MD with long-ranged forces. Matrix multiplication and many quantum chemistry problems, with a similar ratio of communication to computation, are also promising candidates for this architecture. We therefore demonstrated the usability of QCDOC to applications beyond Lattice QCD.

2. The switchless 24-fold parallelism of the QCDOC nearest neighbor network may be a design paradigm to allow scalable performance for future generations of supercomputers.

8. Acknowledgement

Stony Brook undergraduate student Yu Liu helped with the curve fitting and graduate students Jun Zhang and Sriram Krishnamurthy participated in discussions of numerical experiments.

9. References

1. D. Chen, N. H. Christ, C. Cristian, Z. Dong, A. Gara, K. Garg, B. Joo, C. Kim, L. Levkova, X. Liao, R. D. Mawhinney, S. Ohta, and T. Wettig, hep-lat/0011004.

2. IBM Blue Gene Team, Blue Gene: A vision for protein science using a petaflop supercomputer. IBM Systems, **40** (310-327 (2001).

3. M. P. Allen and D. J. Tildesley, Computer Simulation of Liquids, Oxford, NY, 1987.

4 D. Frenkel and B. Smit, Understanding Molecular Simulation, Academic, San Diego, 1996.

5 S. Pfeiffer, D. Fushman, and D. Cowburn, J. Am. Chem Soc. **123**, 3021-3026 (2001)

6. K. Kadau, T. C. Germann, P. S. Lomdahl, and B. L. Holian, Science **296**, 1681 (2002).

7. E. Tajkhorshid, P. Nollert, M. O. Jensen, L. J. W. Miercke, J. O'Connell, R. M. Stroud, and K. Schulten, Science, 296, 525 (2002).

8. M. Samaras, P. M. Derlet, H. Van Swygenhoven, and M. Victoria, Phys. Rev. Lett. **88**, 125505 (2002).

9. M. I. Baskes, Phys. Rev. **B 46**, 2727 (1992).

10. R. Alan McCoy and Yuefan Deng, Int. J. High Performance Computing Applications, **13**, 16 (1999).

11. D. Brown, H. Minoux, and B. Maigret, Computer Physics Commun., **103**, 170 (1997).

12. L. Kale et al., J. Comput. Phys. **151**, 283 (1999).

13. E. Tajkhorshid, P. Hollert, M. O. Jensen, L. J. W. Mierke, J. O'Connell, R. M. Stroud, and K. Schulten, Science **296**, 525 (2002); J. Phillips, G. Zheng, S. Kamur, and L. Kale, Preprint.

14. K. Esselink, Computer Phys. Commun. **87**, 375 (1995).

15. R. Zhou, E. Harder, H. Xu, and B. J. Berne, J Chem Phys., **115**, 2348 (2001).

16. R. W. Hockney and J. W. Eastwood, Computer Simulation Using Pasticles, IOP Bristol, 1988.

17. T. Darden, D. York, and L. Pedersen, J. Chem. Phys. **98**, 10089 (1993).

18. M. Desemo and C. Holm, J. Chem. Phys. **109**, 7678 (1998).

19. M. Desemo and C. Holm, J. Chem. Phys. **109**, 7696 (1998).

20. J. Phillips, G. Zheng, and L. Kale. Proceedings of the IEEE/ACM SC002 Conference (2002), Baltimore MD

21. Y. Duan and P. Kollman, Science **282**, 740 (1998) .

Authors' Brief Biography:

Xiaodan Cai obtained a Ph.D. in Mechanical Engineering from the Stony Brook University in 1997. After a brief stay at the Center of Turbulence Research at Stanford University as a postdoctoral fellow, he joined the Thaerocomp Corporation on Long Island, New York, as a senior researcher and worked there for two years. In 2001, he was appointed by the University of Science and Technology of China as a Professor. His main research interests are parallel computing, turbulent mixing, and micro-reaction systems.

James Davenport received his Ph.D. in Physics from The University of Pennsylvania in 1976. He has been at Brookhaven National Lab since 1978, serving as Associate Chair of the Physics Department, Chair of the Department of Applied Science, and currently as Associate Director of the Center for Data Intensive Computing. His research has involved the use of advanced computing to explore the properties of materials, particularly metals and alloys, solid surfaces, and magnetic systems, using density functional theory and first principles molecular dynamics.

Yuefan Deng obtained a Ph.D. in Theoretical Physics from Columbia University in 1989. After a brief stay at the Courant Institute of NYU as a research associate, he joined the Stony Brook University's applied mathematics faculty where he was promoted to professor of applied mathematics in 1998. He spent one year at the IBM T. J. Watson Research Center as visiting researcher in 1999. His main research interests are parallel computing, molecular dynamics, and their applications to physics and biology.

James Glimm obtained a Ph. D. from Columbia University in 1959. After holding academic positions at MIT, the Courant Institute, NYU, and Rockefeller University, he has been Chair of the Department of Applied Mathematics and Statistics at Stony Brook University and the Director of the Center of Data Intensive Computing, BNL. His research interests are in computational fluid dynamics, partial differential equations, and in turbulence modeling. A recent interest is in molecular dynamics.

Eunice E. Santos obtained a Ph.D. in Computer Science for the University of California, Berkeley in 1995. She also received B.S. and M.S. degrees in both Mathematics and Computer Science. From 1995-2000, Dr. Santos was a faculty member in the Department of Electrical Engineering and Computer Science at Lehigh University. Since 2000, she is an Associate Professor in the Department of Computer Science at Virginia Polytechnic Institute and State University. Her research interests include: parallel and distributed processing, numerical and scientific computing, networking and scheduling, algorithms and complexity, and evolutionary computing. She is the recipient of an NSF CAREER grant.