A Communication-Optimal N-Body Algorithm for Direct Interactions

Michael Driscoll*[1], Evangelos Georganas*[1], Penporn Koanantakool*[1], Edgar Solomonik*, and Katherine Yelick*†
*Computer Science Division, University of California, Berkeley
†Lawrence Berkeley National Laboratory, Berkeley, CA
{driscoll,egeor,penpornk,solomon,yelick}@cs.berkeley.edu

***Abstract*—We consider the problem of communication avoidance in computing interactions between a set of particles in scenarios with and without a cutoff radius for interaction. Our strategy, which we show to be optimal in communication, divides the work in the iteration space rather than simply dividing the particles over processors, so more than one processor may be responsible for computing updates to a single particle. Similar to a *force decomposition* in molecular dynamics, this approach requires up to $\sqrt{p}$ times more memory than a *particle decomposition*, but reduces communication costs by factors up to $\sqrt{p}$ and is often faster in practice than a particle decomposition [1]. We examine a generalized force decomposition algorithm that tolerates the memory limited case, i.e. when memory can only hold c copies of the particles for $c = 1, 2, ..., \sqrt{p}$. When $c = 1$, the algorithm degenerates into a particle decomposition; similarly when $c = \sqrt{p}$, the algorithm uses a force decomposition. We present a proof that the algorithm is communication-optimal and reduces critical path latency and bandwidth costs by factors of c^2 and c, respectively. Performance results from experiments on up to 24K cores of Cray XE-6 and 32K cores of IBM BlueGene/P machines indicate that the algorithm reduces communication in practice. In some cases, it even outperforms the original force decomposition approach because the right choice of c strikes a balance between the costs of collective and point-to-point communication. Finally, we extend the analysis to include a cutoff radius for direct evaluation of force interactions. We show that with a cutoff, communication optimality still holds. We sketch a generalized algorithm for multi-dimensional space and assess its performance for 1D and 2D simulations on the same systems.**

***Keywords*-parallel algorithms; communication-avoiding algorithms; particle methods;**

I. Introduction

Communication time can comprise a significant portion of execution time for parallel algorithms. To make matters worse, hardware trends indicate that interconnection network performance will lag far behind computational performance in exascale machines. One approach to bridging the growing performance gap is the design of new algorithms that provably minimize communication. In this paper, we give such an algorithm for the N-body problem. Using a framework developed by Ballard et al. [2], we give a lower bound on the communication in terms of the number of messages and words sent along the critical path. We show that our algorithm meets the lower bound, and consequently is communication-optimal. Furthermore, we can exploit a "lower" lower bound by taking advantage of extra memory. With enough memory for c copies of the particles, we can theoretically reduce bandwidth and latency costs by factors up to c and c^2, respectively. This effect is also observable in practice: performance results from two systems indicate that our algorithm achieves nearly perfect strong scaling with the right choice of c. We find that maximizing c may not yield the best performance in practice because collectives fail to scale logarithmically as our model assumes, so c should be treated as a tuning parameter.

The contributions of this paper are:

- An interpretation of the lower bound on communication for an N-body simulation timestep. The bound captures both the number of messages and the number of words sent along the critical path.
- An algorithm for particle interactions that achieves the lower bound for a fixed memory size. The algorithm allows finite cutoff distances beyond which interactions have constant or zero effect.
- Performance results showing attainable speedups from the new algorithm on two distributed-memory machines.

The paper is organized as follows. Section II introduces communication-avoiding algorithms and provides a derivation of the communication lower bound for the N-body problem. Section III gives the communication-avoiding N-body algorithm, proves its optimality, and presents performance results from experiments on two supercomputers. Section IV generalizes the algorithm to include a cutoff radius beyond which particles have constant effect. Again, we give a proof of optimality and performance results from real systems. Section V concludes by describing future work.

II. Previous Work

Our communication-avoiding algorithm was motivated by an examination of communication lower bounds for the N-body problem. A recent paper [2] introduced general communication lower bounds applicable to an variety of linear algebra operations. The analysis of these lower bounds has led to new algorithmic developments, in particular

[1]Michael, Evangelos, and Penporn contributed equally to this work and are listed alphabetically.

2.5D algorithms [3]. These communication lower bounds are easily extensible to direct N-body force calculations, as is the concept of 2.5D algorithms. In this section, we detail this motivating work and review related literature and applications which have applied similar analyses to the N-body problem.

A. Communication lower bounds

In 1981, Hong and Kung introduced communication lower bounds for sequential matrix multiplication [4]. These lower bounds were extended to parallel matrix multiplication by Irony et al. [5] and, for the PRAM model, by Aggarwal et al. [6]. More recently, these lower bounds have been generalized to a larger class of dense linear algebra problems by Ballard et al. [2]. The general form of the lower bounds on latency S and bandwidth W can be written in terms of the memory size M, number of operations required by the problem F, and an upper bound on the number of operations that can be done with M operands, H,

$$S = \Omega\left(\frac{F}{H}\right), \qquad W = \Omega\left(S \cdot M\right) = \Omega\left(\frac{M \cdot F}{H}\right). \quad (1)$$

The number of force evaluations that can be computed with M particles can be upper-bounded as $H_{\text{MD}} = O(M^2)$. This upper bound, which represents the maximal amount of potential data-reuse, yields the communication lower bounds

$$S_{\text{MD}} = \Omega\left(\frac{F}{M^2}\right), \qquad W_{\text{MD}} = \Omega\left(\frac{F}{M}\right).$$

If all interactions of n particles are computed on p processors in a load-balanced fashion, each processor must compute $O(n^2/p)$ force evaluations, yielding the following communication lower bounds

$$S_{\text{direct}} = \Omega\left(\frac{n^2}{p \cdot M^2}\right), \qquad W_{\text{direct}} = \Omega\left(\frac{n^2}{p \cdot M}\right). \quad (2)$$

Immediately, we notice that M is in the denominator of these lower bounds, suggesting that increased memory size allows less communication. In the parallel case, it turns out that given extra memory, data-replication can be used to lower the communication cost. In fact, many existing N-body algorithms already use data replication, as we detail below. The novelty of our algorithm is that we parameterize the number of data copies and minimize communication for any amount of memory, as done by 2.5D algorithms for dense linear algebra. 2.5D algorithms are a memory-aware extension of 3D algorithms, which use $p^{1/3}$ copies of data on p processors [7], [8], [6], [9], [10].

B. Particle and force decompositions

The naive approach for parallelizing N-body simulations is to assign each processor n/p particles. If a processor computes all the interactions for its n/p particles with all other particles, the amount of communication required along the critical path is

$$S_{\text{particle}} = O(p), \qquad W_{\text{particle}} = O(n),$$

since each processor must send its data to each other processor.

Plimpton pointed out that by assigning each processor a block of force interactions rather than particles, communication is reduced [1]. In particular, n^2 total interactions need to be computed, and each of p processors computes an $n/\sqrt{p}$-by-$n/\sqrt{p}$ block of the interactions. Thus, each processor requires only $2n/\sqrt{p}$ particles to compute its interactions and must return $2n/\sqrt{p}$ force contributions. Assuming the particle locations are not initially replicated and the forces must be collected at the end, a broadcast and a reduction is required to communicate these data sets. Thus the total communication costs are

$$S_{\text{force}} = O(\log(p)), \qquad W_{\text{force}} = O(n/\sqrt{p}).$$

Ignoring the $\log(p)$ factor, we note that with respect to the particle decomposition, force decomposition reduces latency by a factor of p and bandwidth by a factor of $\sqrt{p}$. However, the memory usage goes up by a factor of $\sqrt{p}$ since each particle is replicated $\sqrt{p}$ times. We note that this memory usage and costs are in accordance with our lower bound, Equation 2.

C. Cutoff and spatial decomposition

In molecular dynamics simulations, it is common to impose a cutoff on direct force interaction evaluations. Force interactions decay with distance, so their evaluation can be truncated by ignoring interactions between particles which lie beyond some cutoff distance. Typically, a correction term accounts for the contribution of long-distance interactions. Since the long-distance contribution to the potential is smooth, grid-based solvers are often employed to evaluate this correction. Instead of analyzing the costs associated with computing long-range interactions, we will instead focus on the analysis of direct interactions within the cutoff distance.

Our lower bounds extend trivially to the case where direct interactions are truncated within a cutoff. The only modification to the argument is a difference in the total number of computations which is now not n^2, but rather $F = nk$, where k is the number of interactions necessary for each particle. The lower bounds on the number of messages and words that must be sent are

$$S_{\text{cutoff}} = \Omega\left(\frac{nk}{p \cdot M^2}\right), \qquad W_{\text{cutoff}} = \Omega\left(\frac{nk}{p \cdot M}\right). \quad (3)$$

A spatial decomposition is a natural choice for parallelizing problems with a cutoff distance. In such a decomposition, each processor owns all particles in a region of the physical simulation domain. Consequently, processors must only communicate with their neighbors, with the number of

neighbors given by the ratio of the cutoff with respect to the length of the simulation box and the dimensionality of the problem space. For instance, given some cutoff $r_c = ml_p$ where l_p is the diameter of the box of particles owned by a processor and m is the number of boxes spanned, each processor must communicate with $O(m^d)$ processors, where d is the number of dimensions. If processors form pairs to compute their particles interaction, a message of size $O(n/p)$ is required between each pair of interacting processors. This spatial decomposition algorithm has a communication cost of

$$S_{\text{spatial}} = O(m^d), \qquad W_{\text{spatial}} = O(nm^d/p).$$

If we plug in $k = O(nm^d/p)$ into Equation 3, we see that the spatial decomposition is communication optimal in the case of minimal memory $M = O(n/p)$.

D. Neutral territory methods

Force decomposition can be done when a cutoff is imposed on interactions, but physical locality must now be considered in the algorithm. Hybrids between force and spatial decomposition can provide communication optimality and yield the methods most commonly used in practice currently. Generally these methods can be defined as 'neutral territory' (NT), since force interactions are computed on processors which own neither of the interacting particles in their assigned spatial territory. These algorithms achieve the communication costs

$$S_{\text{NT}} = O(1), \qquad W_{\text{NT}} = O(nm^d/p^{1.5}),$$

which is asymptotically optimal for $M = O(n/\sqrt{p})$ according to the lower bound (Equation 3).

Snir proposed a hybrid between a spatial and a force decomposition for cutoff interactions in 3-dimensional space in [11]. Snir's algorithm performs a multicast of particle locations to a set of nearby processors. Snir also gave lower bounds on the communication cost for this problem which showed the optimality of his algorithm asymptotically. However, Snir did not consider the limited-memory scenario.

The neutral territory method [12] was introduced by Shaw independently of Snir and achieves the same asymptotic cost. Shaw's algorithm selects a 3-dimensional import region that has a volume smaller than that of Snir's by a constant factor.

The midpoint method [13] is another interesting variation of neutral territory methods. In the midpoint method, a processor computes all interactions for which the midpoint of the interacting particles lies in the processor's territory. While the method is not asymptotically optimal, it has a smaller import region for a typical number of processors. The method also has advantages in latency and throughput on torus networks, and can be generalized to multi-particle interactions.

E. Existing applications

Existing molecular dynamics applications that employ neutral territory decompositions already utilize the knowledge that replication can lower communication cost.

NAMD [14] is a parallel molecular dynamics simulation package which runs on top of the Charm++ runtime system [15]. Charm++ is an object-based parallel framework which decomposes work and data into 'chare' objects. Arrays of chare objects are dynamically mapped to processors by the runtime system. Chares communicate with each other via asynchronous message invocations.

NAMD employs an algorithm that decomposes particles spatially into an array of 'patch' chare objects and decomposes forces into 'compute' chare objects. Patches communicate particle data to compute objects and compute objects force contributions back to patches. This algorithm employs data replication, and achieves the same asymptotic communication complexity as the neutral territory algorithm from the previous section.

The neutral territory method as described by Shaw [12] is employed by the specialized supercomputer Anton [16]. This supercomputer is designed to run parallel molecular dynamics simulations at high efficiency. The architecture was co-designed with consideration for the 3D structure of the spatial decomposition and the neutral territory interaction algorithm.

Desmond [17] is a general-purpose molecular dynamics package developed in association with Anton. Desmond uses the midpoint method to evaluate direct interactions. This software achieves good absolute performance and scalability on modern architectures such as the BlueGene/P.

III. Interactions with No Cutoff Radius

This section addresses simulations in which every particle interacts with every other particle. We present a communication-optimal algorithm for computing interactions between all pairs of particles, provide a proof of its optimality, and present performance results from our implementation.

A. The all-pairs interaction algorithm

The communication-avoiding algorithm, given in Algorithm 1 as pseudocode, uses p processors to compute interactions between a set S of n particles. The algorithm also takes as input a replication factor c describing the number of times the set of particles can be replicated in available memory. The processors are arranged in a two-dimensional grid with p/c columns (or teams) and c rows. Particles are distributed evenly among the teams into local subsets S_t (for team t), and teams are responsible for computing updates to their local subset.

The algorithm proceeds as follows. A team leader broadcasts S_t to the rest of the team. Each team member makes a second copy in an exchange buffer. Each processor then

Algorithm 1 S' = CA-ALL-PAIRS-N-BODY(S, c)

Input: Replication factor c, the number of extra copies of the particles that will fit in memory.
Input: p processors arranged in two-dimensional grid of p/c columns (teams) and c rows.
Input: Set S of n particles divided evenly among team leaders into local subsets S_t.
Output: Set S' of n particles where every particle has interacted with every other particle.

1: `// In parallel on all processors:`
2: Broadcast S_t from team leader to team members.
3: Copy S_t to exchange buffer S'_t of size $\lceil nc/p \rceil$.
4: Given a k^{th}-row processor, shift S'_t by k along row.
5: **for** p/c^2 **steps do**
6: Shift S'_t by c along row.
7: Update particles in S_t based on effect of S'_t.
8: **end for**
9: Sum-reduce updates within team.

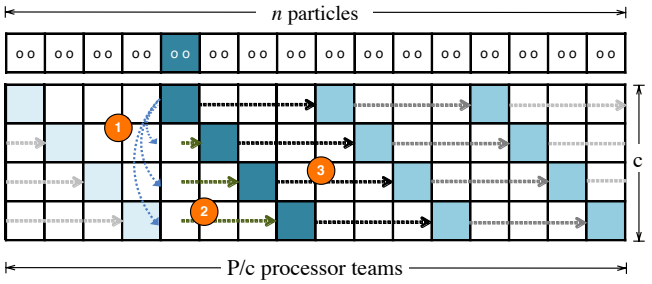


Figure 1: Illustration of Algorithm 1. The color boxes traces the path of the fifth team's particles. The labels show: (1) the initial broadcast step within a team; (2) the skewing of particles according to the row index; (3) the first of p/c^2 shifts and updates. Not shown is a final reduction within a column to combine updates.

shifts its exchange buffer row-wise, with the shift magnitude determined by the row index (i.e. processors on row four shift east by four modulo p/c). Then, for p/c^2 steps, each processor shifts its exchange buffer by c and, upon receiving a new set of particles, updates S_t accordingly. Finally, sum-reductions within each team combine the updates. Figure 1 illustrates the algorithm.

Our algorithm "interpolates" between the particle decomposition and force decomposition algorithms of Plimpton [1]. In fact, either decomposition falls out at extreme values of c. When $c = 1$, the algorithm resembles a particle decomposition with simple point-to-point shifting, and each team of one processor is responsible for computing all forces on its subset of particles. Similarly, when $c = \sqrt{p}$, the algorithm uses a force decomposition.

B. Communication optimality

Recall from Equation 2 the lower bounds for communication in a direct N-body timestep,

$$S_{\text{MD}} = \Omega\left(\frac{n^2/p}{M^2}\right), \qquad W_{\text{MD}} = \Omega\left(\frac{n^2/p}{M}\right).$$

An analysis of our algorithm reveals that it meets these bounds.

By definition, the replication factor c is the number of extra copies of the particles than can be stored in memory. It follows that the total memory required per core is

$$M = O\left(c \cdot \frac{n}{p}\right). \tag{4}$$

We analyze the communication along the critical path to ascertain a lower bound on the total communication cost. The cost can be expressed as the sum of costs of three phases: the initial broadcast and skewing, the shifting steps, and the final reduction. First, each of p/c teams executes a broadcast of $O(cn/p)$ words among c processors. Assuming logarithmic collective performance, the broadcast can be completed in parallel with $\log(c)$ messages. Then, each processor skews its particles row-wise in parallel, sending $O(cn/p)$ words in $O(1)$ messages. Next, the processors perform p/c^2 shifting steps in which they send $O(1)$ messages of $O(cn/p)$ words, yielding a subtotal cost of $O(p/c^2)$ messages and $O(n/c)$ words. A final reduction moves $O(cn/p)$ words in $\log(c)$ messages. Asymptotically, the total cost is

$$S_{ca} = O\left(\frac{1}{c^2} \cdot p\right), \qquad W_{ca} = O\left(\frac{1}{c} \cdot n\right). \tag{5}$$

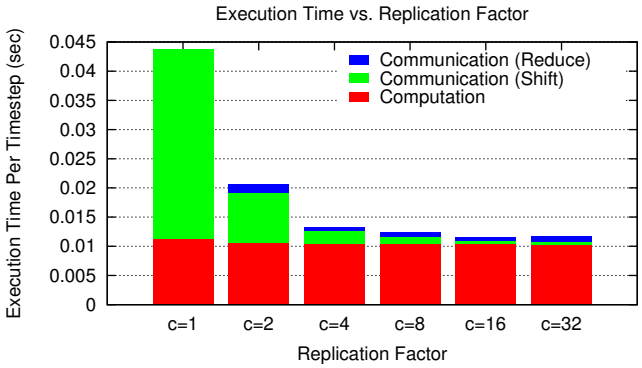
Solving Equation 4 for c and substituting the result reveals that the algorithm meets the lower bounds:

$$S_{ca} = O\left(\frac{n^2/p}{M^2}\right), \qquad W_{ca} = O\left(\frac{n^2/p}{M}\right).$$

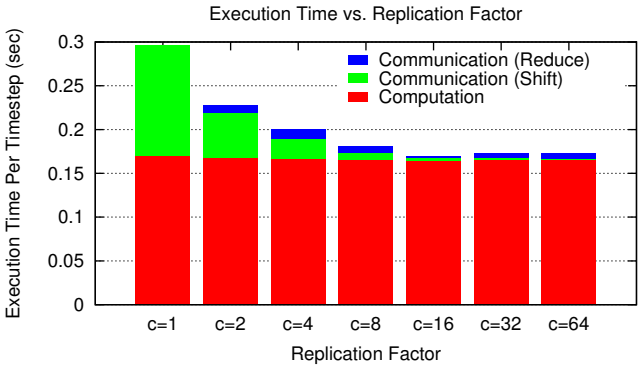
This analysis reveals that the algorithm is communication-optimal for all integral values of $c = 1, 2, ..., \sqrt{p}$. However, observe that the lower-bound itself is a function of c. Surprisingly enough, we can realize a "lower" lower-bound by utilizing more memory. This is the key insight into why our algorithm reduces communication in practice.

C. Performance results

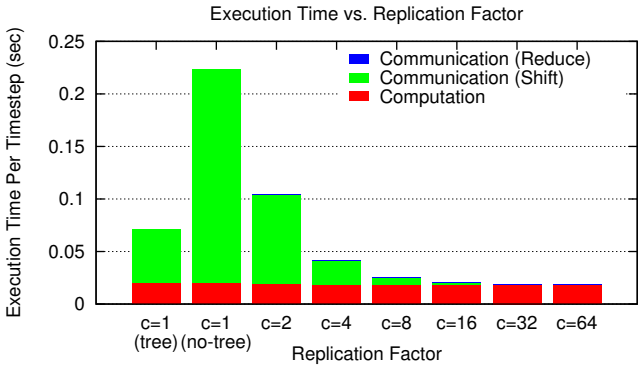
We built a simple particle simulation that uses our algorithm and measured its performance on two machines from different vendors. Our code simulates particles moving in a two-dimensional space with reflective boundary conditions. The particles exert a repulsive force on each other that drops off with the square of their distance. The force is symmetric, but it need not be and we do not apply optimizations to exploit the symmetry. The particles are 52 bytes in size.



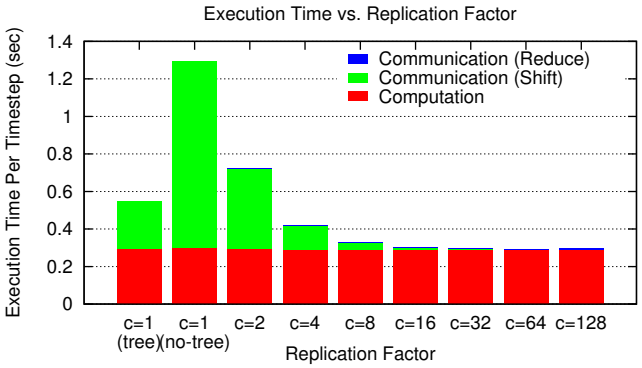
(a) Hopper, 6,144 cores, 24,576 particles.



(b) Hopper, 24,576 cores, 196,608 particles.



(c) Intrepid, 8,192 cores, 32,768 particles.



(d) Intrepid, 32,768 cores, 262,144 particles.

Figure 2: The effect of the replication factor c on execution time for small and large problems on Hopper and Intrepid. Figure 2a shows monotonically decreasing communication with increasing c, as predicted by the model. In contrast, Figure 2b shows best performance when $c = 16$; this is the point at which the communication pattern strikes a balance between collective and point-to-point costs. Similar are the conclusions for Intrepid on Figures 2c and 2d.

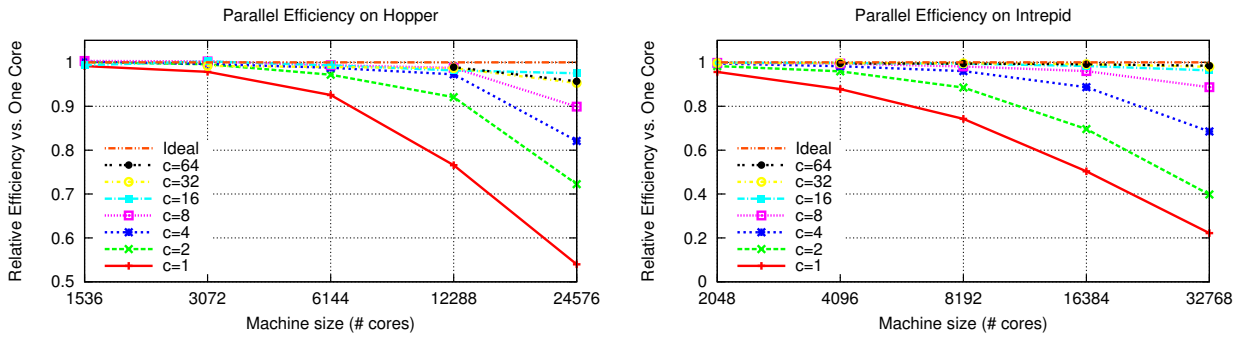
We ran our experiments on two platforms, Hopper and Intrepid. Hopper [18] is a 6,384 node Cray XE-6 machine located at NERSC. It's currently ranked #19 on the Top500 list [19]. Each Hopper node has two 12-core, AMD MagnyCours processors running at 2.1 GHz, yielding 24 cores per node and 153,216 cores in total. [1] Nodes are connected in a three-dimensional torus via the Cray Gemini interconnect. Intrepid [20] is a 163,480 core IBM BlueGene/P machine located at ALCF. Each Intrepid node consists of one quad-core, 850 MHz PowerPC processor connected in a three-dimensional torus. Intrepid currently ranks #47 on the Top500 list. All codes were developed in C using MPI.

Since Intrepid provides topology-aware partitions, we modified the code to utilize topology-aware collectives provided by the DCMF communication layer [21]. In doing so, we were able to fully exploit the bidirectional network links and minimize network contention. We found that replacing P/c^2 point-to-point shifts within the rows with P/c^2 broadcasts across the rows improved performance because the bidirectionality of the torus provides twice the bandwidth of a point-to-point send along a single link.

Our experiments sought to understand: 1) the effect of increased replication factors for fixed machine sizes and problem sizes, and 2) the strong scaling performance for all replication factors across a fixed problem size.

1) Scaling the replication factor: The model predicts that communication cost should drop by factors between c and c^2 for increased c. In practice, we find this to be accurate for small c. Figure 2 shows the breakdown of communication and computation time for 24K-particle and 196K-particle simulations on 6K and 24K cores of Hopper, respectively. When $c = 1$, communication costs represent significant portions of execution time. As c increases, we see communication costs more-than-halving until $c = 16$. When $c = 64$ in the larger simulation, we see a greater cost than when $c = 16$, even though the model predicts better

[1] Given Hopper has 24 cores per node, runs that saturate all cores often have factors of 3 in them that make our choice of experimental parameters seem odd. Powers-of-two numbers can be recovered by dividing by 3 in most cases.



(a) Hopper, 196,608 particles.

(b) Intrepid, 262,144 particles

Figure 3: Strong scaling performance on Hopper and Intrepid. For the given problem sizes, our algorithm achieves nearly perfect strong scaling with the appropriate choice of replication factor.

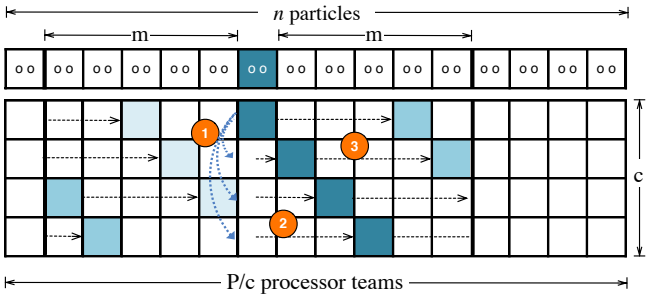


Figure 4: An illustration of Algorithm 2, the communication-avoiding algorithm for distance-limited interactions. Labels (1) and (2) show the initial broadcast within a team and row-wise skewing. Label (3) shows the first of $2m/c$ shifts that "wrap around" at the cutoff radius.

performance for the pure force decomposition. We believe the communication pattern at this point best balances the costs of collective and point-to-point communication. We do not plot the initial broadcast cost because it is negligible.

Figures 2c and 2d show the execution time breakdown for 32K-particle and 262K-particle simulations on 8K and 32K cores of Intrepid, respectively. Note that we plot two runs for $c = 1$. The *tree* bar represents a run which utilized a special network for collective operations involving the whole partition. Alternatively, we forced the use of the regular 3D-torus for the *no-tree* run. The specialized network is effective for the naive implementation of the interaction algorithm, but our algorithm eventually outperforms the hardware-assisted variant by using the torus intelligently. For runs that just use the torus, we see a 99.5% reduction in communication time.

2) Strong scaling performance: We ran additional experiments to assess the strong scaling performance of the algorithm. Figure 3 shows the data from 196K and 262K particle runs on Hopper and Intrepid, respectively. Our algorithm achieves nearly perfect strong scaling with the right choice of c.

Algorithm 2 S' = CA-1D-N-BODY(S, r_c, c)

Input: The number of processors m spanned by the cutoff distance. m implicitly includes r_c via Equation 6.
Input: Replication factor c, the number of extra copies of the particles that will fit in memory.
Input: p processors arranged in two-dimensional grid of p/c columns (teams) and c rows.
Input: Set S of n particles divided spatially among team leaders into local subsets S_t.
Output: Set S' of n particles where every particle has interacted with every other particle.

1: `// In parallel on all processors:`
2: Broadcast S_t from team leader to team members.
3: Copy S_t to exchange buffer S'_t of size $\lceil nc/p \rceil$.
4: Given a k^{th}-row processor, shift S'_t by k along row modulo the cutoff window.
5: **for** $2m/c$ **steps do**
6: Shift S'_t by c along row modulo the cutoff window.
7: Update particles in S_t based on effect of S'_t.
8: **end for**
9: Sum-reduce updates within team.

IV. Finite Cutoff Distance

The communication-avoiding algorithm can be generalized to the case where particles have no effect beyond a cutoff radius r_c, or their effect can be approximated by a constant value. Here, we describe the algorithm in one-dimensional space, explain how it can be generalized to higher dimensional spaces, show that communication-optimality still holds, and give performance results from 1D and 2D experiments on Hopper and Intrepid.

A. The 1D interaction algorithm

The algorithm for distance-limited interactions in one dimension can be described as the algorithm for interactions with no cutoff plus two key refinements. First, we assume

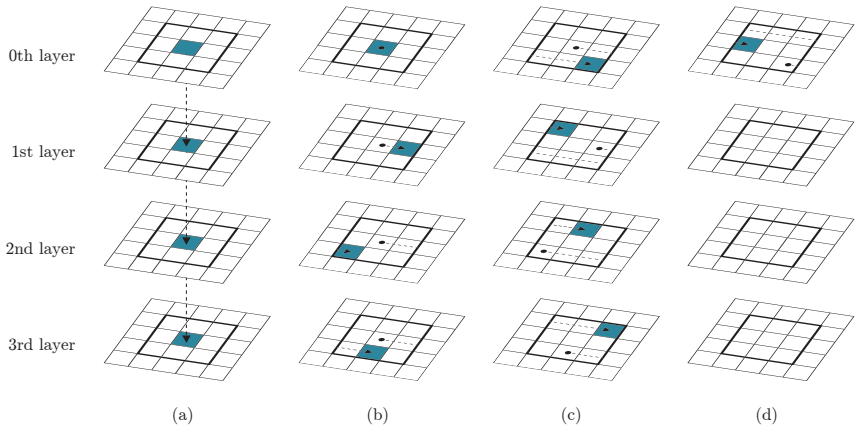


Figure 5: Illustration of the communication avoiding algorithm for distance limited interactions in two dimensions. This example shows 100 processors arranged into 25 teams and 4 replication layers. The arrows indicate the transfer of particles between processors for different steps: (a) Broadcast among team members. (b) Initial skew. (c) Shifts during the second iteration. (d) Shifts during the third (final) iteration.

a spatial decomposition of particles among teams, i.e. each team is responsible for the particles in a particular region of the simulation space. As before, we assume a uniform particle distribution for load balance. Second, the algorithm performs shifts modulo the cutoff distance, not the edge of the simulation space as before. For purpose of understanding, we translate the cutoff radius into the number of processors m spanned by the radius in a simulation space of length l:

$$\frac{r_c}{l} = \frac{mc}{p}. \tag{6}$$

Figure 4 illustrates the algorithm for simulations in 1D space, and Algorithm 2 gives pseudocode.

B. Optimality of the 1D interaction algorithm

In the presence of a cutoff radius, the computational cost along the critical path decreases relative to the all-pairs interaction algorithm. Specifically, assuming uniform particle distribution, the number k of necessary interactions per particle is

$$k = \frac{2r_c}{l}n = O\left(\frac{mc}{p}n\right) \tag{7}$$

and the total memory required per processor is

$$M = O\left(c \cdot \frac{n}{p}\right). \tag{8}$$

Analysis of the communication cost along the critical path is similar to that of the all-pairs interaction algorithm. The initial broadcast sends $O(cn/p)$ words to c processors using $\log(c)$ messages, and the initial skewing sends $O(1)$ messages of size $O(cn/p)$ words. This algorithm proceeds with $O(m/c)$ shifting steps in which each processor sends $O(1)$ message of $O(cn/p)$ words, resulting in a total cost of $O(m/c)$ messages and $O(mn/p)$ words. The final reduction sends $O(cn/p)$ words in $\log(c)$ messages. The total communication costs of the algorithm are

$$S_{1D} = O\left(\frac{m}{c}\right), \qquad W_{1D} = O\left(\frac{mn}{p}\right).$$

Using equations 7 and 8, we can rewrite the costs as

$$S_{1D} = O\left(\frac{nk}{pM^2}\right), \qquad W_{1D} = O\left(\frac{nk}{pM}\right).$$

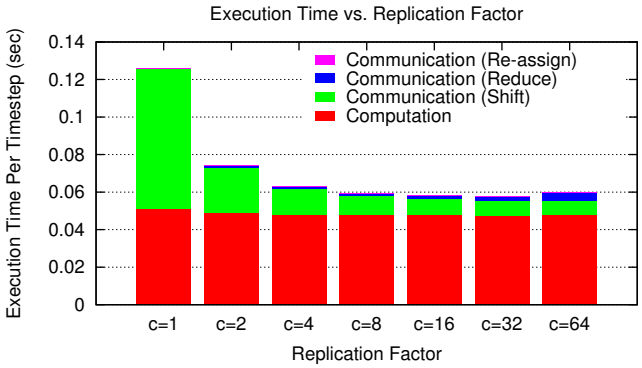
These costs meet the lower bounds from equation 3, hence the algorithm is communication-optimal for all values of $c = 1, 2, ..., m$.

C. Generalization for higher dimensional spaces

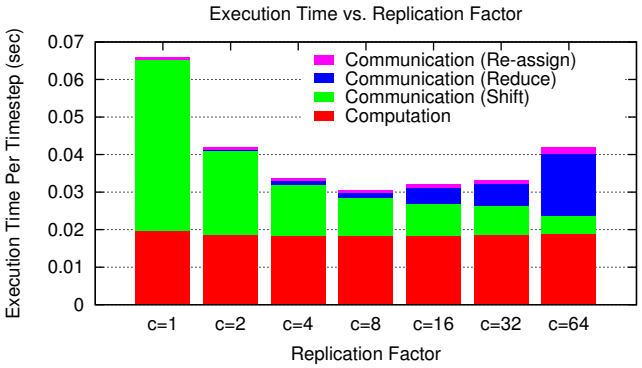
As in the 1D case, we assume a spatial decomposition of particles among processors in a grid of the same dimensionality. We again add a replication dimension of length c. Broadcasts and reductions still occur along the c dimension, and shifts of magnitude c occur in the hyperplane perpendicular to the c dimension. It is difficult to visualize shifts occurring through multiple dimensions, so we recommend linearizing the high-dimensional space, calculating shifts in 1D, and mapping the pattern back into the original space. Figure 5 shows shifts through a 2D space with 25 teams and a replication factor of 4. Communication avoidance becomes especially important in higher dimensions because the number of neighbors is exponential in the dimensionality of the problem space.

D. Performance results

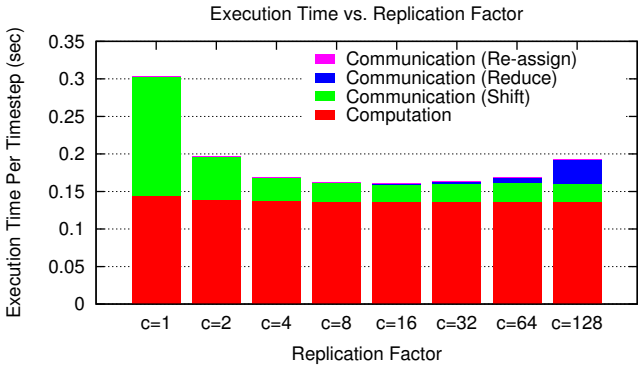
To assess the performance of the generalized algorithm, we extended our codes from Section III with support for



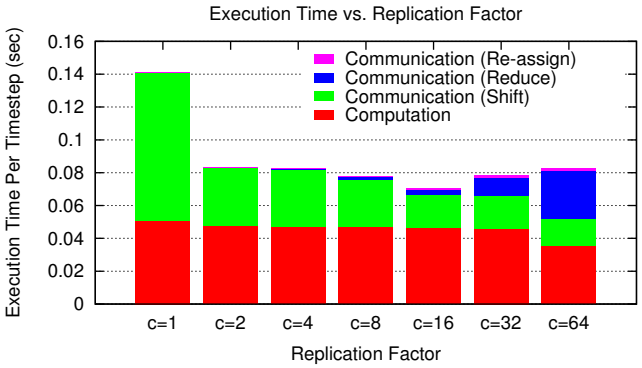
(a) 1D-cutoff, Hopper, 24,576 cores, 196,608 particles.



(b) 2D-cutoff, Hopper, 24,576 cores, 196,608 particles.



(c) 1D-cutoff, Intrepid, 32,768 cores, 262,144 particles.



(d) 2D-cutoff, Intrepid, 32,768 cores, 262,144 particles.

Figure 6: The effect of increased replication factors on execution time for 1D and 2D simulations with a cutoff radius.

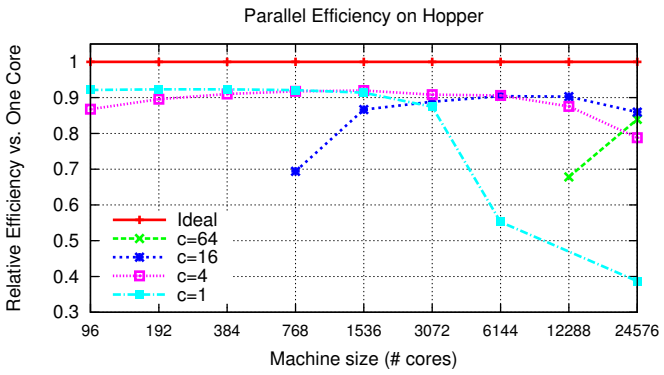
a cutoff radius. We implemented simulations in 1D and 2D space using a spatial decomposition and ran our experiments on the same systems. We also assume a uniform particle distribution, which is fair in a molecular dynamics simulation with a homogeneous environment. Furthermore, we set the parameters of the system to be such that particle distribution remains nearly uniform throughout the simulation. For practicality reasons, we also require that the replication factor "fit inside" the interaction diameter (mathematically: $c \leq 2m$) to ensure all processors in the replication dimension have work to do. We chose the cutoff radius to be $1/4$ of the simulation space to allow reasonably many choices of c. Lastly, we did not utilize Intrepid's topology-aware collectives because the communication pattern did not match the interconnect topology exactly.

1) Scaling the replication factor: We varied c for fixed problem size and machine size to measure the benefit of increased replication factor in 1D and 2D simulations. The results are shown in Figure 6. The additional cost of updating the spatial decomposition at every timestep is labeled as reassignment time in the plot.

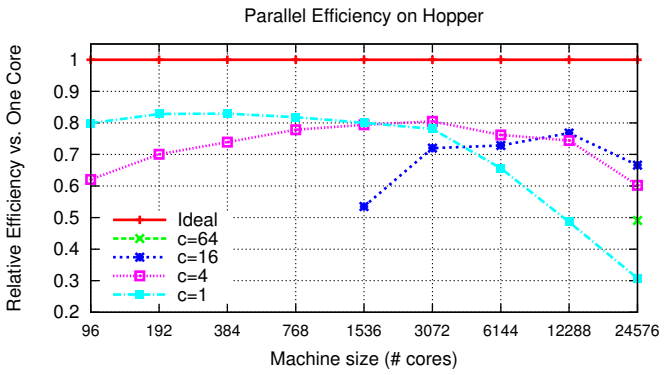
For small values of c, the plots show the expected decrease in communication time. However, for large c the cost of the reduction step grows considerably, yielding poorer overall performance than intermediate c values. We believe this effect is primarily caused by collectives' inability to scale logarithmically when communication teams reach a certain size. Fortunately, our algorithm can be tuned to find the replication factor that provides the best balance between collective and point-to-point performance.

Costs due to shifting appear to stagnate after a few c values, unlike in Section III where they approached zero. We believe this is due to measurable load imbalance as lightly-loaded processors must wait longer for particles to be shifted from heavily-loaded processors. This kind of load imbalance does not occur for interactions between all pairs because a spatial decomposition is not required in that case.

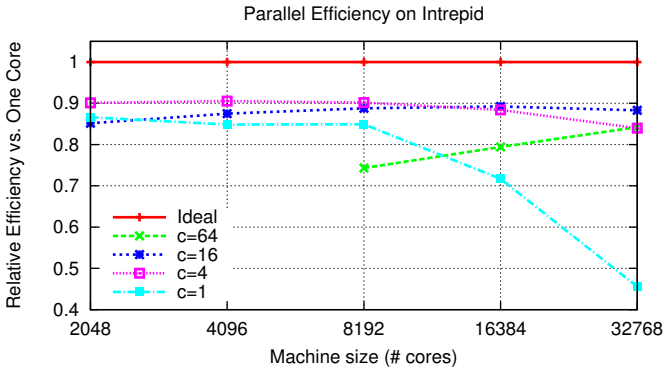
2) Strong scaling performance: Figure 7 shows the strong scaling performance of 1D and 2D simulations with 196K particles on Hopper and 262K particles on Intrepid. From the graphs we make several observations. First, the largest available replication factor never gives best results. Second, there is a general trend that for a given replication factor, the algorithm exhibits sub-optimal performance on smaller machines due to load imbalance. Lastly, although we do not see clear benefits at small scale, the best replication of the



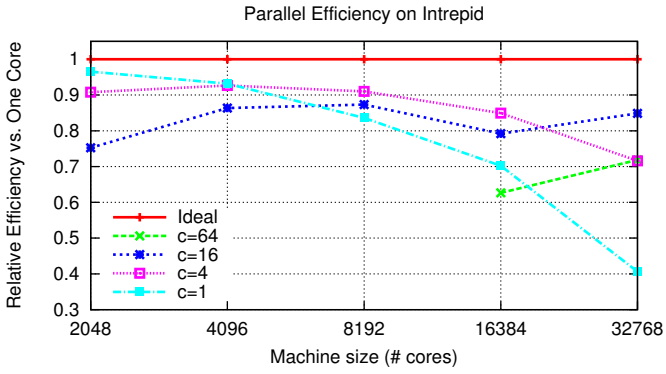
(a) 1D-cutoff, Hopper, 196,608 particles.



(b) 2D-cutoff, Hopper, 196,608 particles.



(c) 1D-cutoff, Intrepid, 262,144 particles.



(d) 2D-cutoff, Intrepid, 262,144 particles.

Figure 7: Strong scaling performance of 1D and 2D simulations with cutoff radius.

communication-avoiding algorithm yields roughly double the efficiency of a non-replicating algorithm on the largest machine sizes (24K cores on Hopper and 32K cores on Intrepid).

In our results, simulations with a cut-off distance exhibit lower parallel efficiency than simulations without a cutoff. We primarily attribute this fact to load imbalance caused by our choice of physical domain decomposition. More specifically, processors assigned to regions near the simulation space boundary have fewer interactions to compute than processors in the middle, leading to increased idle time and critical path length. Secondly, we did not use topology-aware collectives during the shift communication phase on Intrepid; consequently, we utilize only half the bandwidth available to the experiment in Section III because we don't take advantage of the bidirectionality of the torus.

V. Conclusions and Future Work

We have presented an N-body algorithm for direct interactions that uses extra memory to replicate particles and asymptotically reduce communication. We analyzed the lower bounds on communication, and showed that the new algorithm is optimal in communication. We also presented experimental analysis on tens of thousands of cores for both BlueGene/P and Cray XE-6, which show that with the appropriate choice of replication factor, our algorithm achieves nearly perfect strong scaling by striking a balance between point-to-point and collective communication costs. One example shows a speedup of over $11.8\times$ from communication avoidance. While the benefits of communication avoidance are best for small problems on large numbers of cores, the absolute communication overhead for the optimized algorithms is low, resulting in good absolute performance.

While the theoretical analysis would suggest maximizing the amount of replication to $\sqrt{p}$ if memory is available, we found this was not always optimal in practice. We therefore leave as open the question of how to select the replication factor c, which is typically close to the $\sqrt{p}$ limit and can be autotuned at runtime by trying multiple factors. Even using the maximum value of c may be acceptable: for the all-pairs algorithm, the best value of c differed by no more than 16% in any experiment, and most experiments revealed less than 2% difference.

In addition to the specifics of this N-body algorithm, our work represents an application of communication-avoidance

theory beyond numerical linear algebra. This suggests a general strategy for communication avoidance through replication, a technique used previously to increase parallelism or decrease synchronization [22]. We hope that our work prompts similar analyses of other domains and inspires new algorithms that provably minimize communication.

ACKNOWLEDGMENTS

This research used resources of the National Energy Research Scientific Computing Center and the Argonne Leadership Computing Facility, both supported by the Office of Science of the U.S. Department of Energy under Contracts No. DE-AC02-05CH11231 and DE-AC02-06CH11357, respectively. The third author was supported by a Fulbright Scholarship. The fourth author was supported by a Krell Department of Energy Computation Science Graduate Fellowship, grant number DE-FG02-97ER25308.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

REFERENCES

[1] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *J. Comput. Phys.*, vol. 117, no. 1, pp. 1–19, Mar. 1995. [Online]. Available: http://dx.doi.org/10.1006/jcph.1995.1039

[2] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Minimizing communication in linear algebra," *SIAM J. Mat. Anal. Appl.*, vol. 32, no. 3, 2011.

[3] E. Solomonik and J. Demmel, "Communication-optimal 2.5D matrix multiplication and LU factorization algorithms," in *Lecture Notes in Computer Science, Euro-Par, Bordeaux, France*, Aug 2011.

[4] H. Jia-Wei and H. T. Kung, "I/O complexity: The red-blue pebble game," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, ser. STOC '81. New York, NY, USA: ACM, 1981, pp. 326–333.

[5] D. Irony, S. Toledo, and A. Tiskin, "Communication lower bounds for distributed-memory matrix multiplication," *Journal of Parallel and Distributed Computing*, vol. 64, no. 9, pp. 1017 – 1026, 2004.

[6] A. Aggarwal, A. K. Chandra, and M. Snir, "Communication complexity of PRAMs," *Theoretical Computer Science*, vol. 71, no. 1, pp. 3 – 28, 1990.

[7] E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," *SIAM Journal on Computing*, vol. 10, no. 4, pp. 657–675, 1981.

[8] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar, "A three-dimensional approach to parallel matrix multiplication," *IBM J. Res. Dev.*, vol. 39, pp. 575–582, September 1995.

[9] J. Berntsen, "Communication efficient matrix multiplication on hypercubes," *Parallel Computing*, vol. 12, no. 3, pp. 335 – 342, 1989.

[10] S. L. Johnsson, "Minimizing the communication time for matrix multiplication on multiprocessors," *Parallel Comput.*, vol. 19, pp. 1235–1257, November 1993.

[11] M. Snir, "A note on n-body computations with cutoffs," *Theory of Computing Systems*, vol. 37, pp. 295–318, 2004.

[12] D. E. Shaw, "A fast, scalable method for the parallel evaluation of distance-limited pairwise particle interactions," *Journal of Computational Chemistry*, vol. 26, no. 13, pp. 1318–1328, 2005.

[13] K. J. Bowers, R. O. Dror, and D. E. Shaw, "The midpoint method for parallelization of particle simulations," *The Journal of Chemical Physics*, vol. 124, no. 18, p. 184109, 2006.

[14] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kal, and K. Schulten, "Scalable molecular dynamics with namd," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.

[15] L. V. Kale and S. Krishnan, "CHARM++: a portable concurrent object oriented system based on C++," in *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, ser. OOPSLA '93. New York, NY, USA: ACM, 1993, pp. 91–108.

[16] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao, M. P. Eastwood, J. Gagliardo, J. P. Grossman, C. R. Ho, D. J. Ierardi, I. Kolossváry, J. L. Klepeis, T. Layman, C. McLeavey, M. A. Moraes, R. Mueller, E. C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, and S. C. Wang, "Anton, a special-purpose machine for molecular dynamics simulation," in *Proceedings of the 34th annual international symposium on Computer architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 1–12.

[17] K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, J. K. Salmon, Y. Shan, and D. E. Shaw, "Scalable algorithms for molecular dynamics simulations on commodity clusters," 2006.

[18] "Hopper, NERSC's Cray XE6 System." [Online]. Available: http://www.top500.org

[19] "TOP500 Supercomputer Site." [Online]. Available: http://www.top500.org

[20] "Intrepid Guide — Argonne Leadership Computing Facility." [Online]. Available: https://www.alcf.anl.gov/resource-guides/intrepid-and-surveyor-guide

[21] A. Faraj, S. Kumar, B. Smith, A. Mamidala, and J. Gunnels, "MPI collective communications on the Blue Gene/P supercomputer: Algorithms and optimizations," in *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on*, 2009.

[22] K. Madduri, J. Su, S. Williams, L. Oliker, S. Ethier, and K. Yelick, "Optimization of parallel particle-to-grid interpolation on leading multicore platforms," vol. 23, no. 10, 2012.