ROBUST MOLECULAR DYNAMICS SIMULATIONS USING CODED FFT ALGORITHM

Yuk Wong[⋆]*, Yuqiu Zhang*[⋆]*, Haewon Jeong*[†]*, and Pulkit Grover*[†]

[⋆] The Chinese University of Hong Kong, [†]Carnegie Mellon University

ABSTRACT

As error/failure rates in supercomputers are projected to grow, computationally intensive scientific applications that leverage large-scale parallelization will suffer from the increased error rate. In this work, we apply *"coded computing"* to protein folding simulations in an error-prone environment. We implemented the fast Fourier Poisson method for solving electrostatic equations at each time step of the simulation, and we utilize coded FFT algorithm to protect the compute-intensive FFT algorithm from soft errors. Through experiments on Amazon AWS, we showed that coded protein folding can be implemented with less than 10% overhead in total simulation time, and also showed that coded computing approach is faster than classical checkpointing method when the error rate is high.

***Index Terms*—** Molecular Dynamics, Protein Folding, Bioinformatics, Coded Computing, Fast Fourier Transform

1. INTRODUCTION

Recently, there has been growing interest in *"coded computing"* [1–7], in which the core idea is to add redundant computing nodes in a distributed computing system to protect against faulty or straggling nodes. However, the application of coded computing has been mostly limited to machine learning algorithms on cloud systems. In this work, we propose to apply coded computing for scientific computing, specifically protein folding simulations. Molecular dynamics (MD) simulations have been a valuable tool to understand the dynamics of protein folding by providing high spatial and temporal resolution to complement experimental studies [8, 9]. However, simulating protein molecules is challenging due to the sheer size of proteins; they are made up of hundreds of amino acids each containing tens of atoms, and titin, the largest protein in our body, consists of more than 30,000 amino acids [10]. Accurate all-atom MD simulations of proteins became possible only recently due to the advances in parallel computing on high-performance supercomputers [11–14].

While the increased parallelism in a supercomputer enabled large-scale MD simulations, it also reduced the mean time between failures (MTBF) simply because there are more components that can fail [15–17]. To mitigate this issue, algorithmic-based fault tolerance (ABFT) has been proposed

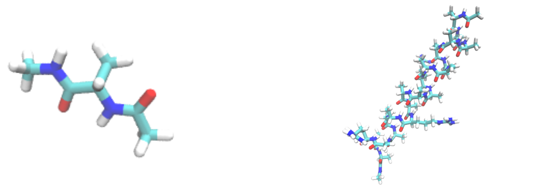

(a) Alanine dipeptide (b) Folded short peptide

Fig. 1. Proteins used for simulation (solvent excluded)

in high-performance computing (HPC) literature [18–21]; the idea resembles coded computing – adding redundant compute nodes by encoding the input data. Such algorithmic-specific fault tolerance techniques can be attractive solutions for future supercomputers for several reasons. First, the traditional checkpoint/restart technique is not scalable since checkpoints are stored at a shared stable memory, and thousands of nodes accessing the shared memory would be extremely slow. Secondly, we expect to see more frequent errors, e.g., every few minutes, and when there is an error, the recovery using ABFT (or coded computing) techniques is orders of magnitude faster than checkpoint/restart methods. This is because there is no need to roll back and recompute the corrupted output using ABFT techniques; we can correct the corrupted output and roll forward. Lastly, it was suggested that increased fault tolerance using ABFT in conjunction with undervolting can be a more energy-efficient solution for exascale computers [22].

In this work, we propose building fault tolerance for parallel MD simulations by applying our recent advance in coded fast Fourier Transform (FFT) algorithm [7]. The most time-consuming and substantial part of MD simulations is the calculation of electrostatic forces between all pairs of atoms, which will take $O(N^2)$ time complexity with a brute-force implementation (N: number of atoms), and it reduces to $O(N \log N)$ using the FFT-based fast Fourier Poisson (FFP) method [23]. We thus limit the scope fault protection to only the FFT computation as the remaining parts consume $O(N)$ time. We implemented *"Coded Protein Folding on AWS"* as a proof-of-concept, and experimented with different numbers of nodes up to 16 nodes. Our experiments show that the overhead of coding is at most $\sim$10% of the total computation time, and the overall performance of the coded algorithm scaled with the number of worker nodes. While our experiments are limited to a small number of nodes, how the algorithm would scale with thousands of nodes remains to be seen.

2. MATHEMATICAL BACKGROUNDS ON MOLECULAR DYNAMICS SIMULATIONS

Molecular dynamics simulates molecular interactions by solving the classical equations of motion in fine-grained time steps. In a simple system with potential energy U, the motion of atoms varying with time t can be formulated as:

$$m_i \frac{d^2 r_i}{dt^2} = f_i, \;\; f_i = -\frac{dU}{dr_i}$$

Here m_i, r_i, f_i are the mass, position and force exerted on particle i, respectively. In common biomolecule force fields, the forces consist of 3 components: bond forces, van der Waals forces, and electrostatic forces. Among them, the bond forces and van der Waals forces decay rapidly with separation and can be safely evaluated with cutoff, resulting in $O(N)$ complexity, where N is the number of atoms in the system. However, the electrostatic energy has a slow decay with distance, and its brute-force calculation would result in O(N^2) complexity. We apply the FFP method in [23] to accelerate the calculation of electrostatic energy, in which the FFT computation is superlinear with O($NlogN$) complexity and make up of most of computational load. Basically, the total electrostatic energy can be written as:

$$E = \sum_{i=1}^{N} \phi_i(\boldsymbol{r_i}) q_i, \tag{1}$$

where $\phi_i(\boldsymbol{r_i})$ is the electrostatic potential at $\boldsymbol{r_i}$ induced by all the other atoms:

$$\phi_i(\boldsymbol{r_i}) = \int \frac{\boldsymbol{\rho(r)}}{\epsilon|\boldsymbol{r} - \boldsymbol{r_i}|} d^3\boldsymbol{r}. \tag{2}$$

Here, $\boldsymbol{\rho(r)}$ represents the charge distribution. Our computation framework is based on the well-known Ewald sum method [24]. Conceptually, since the terms in the original summation formula decay relatively slowly with distance, Ewald sum method *"smooths"* the charge distribution in order to improve computational efficiency, by omitting some terms in long-distance interactions. The classic Ewald sum form is given by:

$$E = \sum_{i=1}^{N}\sum_{j>i}^{N} \frac{q_i q_j (erfc(\frac{r_{ij}}{\epsilon\sqrt{2}\sigma}))}{\epsilon r_{ij}} + \frac{1}{2}\sum_{i=1}^{N} \phi^{\sigma}(\boldsymbol{r_i}) q_i - \frac{\sum_{i=1}^{N} q_i^2}{\epsilon\sqrt{2\pi\sigma^2}}.$$

The Ewald sum method splits the summation into three parts: the first term is the direct summation to which some Gaussian function with standard deviation σ is added. It is therefore made smooth and decays very quickly with the distance r_{ij} due to the nature of erfc function and can hence be omitted once r_{ij} is larger than some pre-defined cutoff radius. The second term deals with the long-range electrostatic interactions between particles, which can be efficiently computed by techniques such as FFT [23], multigrid [25] or FMM [26]. The last term is a constant to compensate for the self-interaction effect induced by the first term. Hence, in terms of computational efficiency, the problem reduces to realizing fast and reliable evaluation of long-range electrostatic interactions, where our coded FFT technique come into use. The overall process is described in more detail in Section 4.

3. SYSTEM MODEL

Error Model: In our model, we assume errors only occur during the most computationally intensive steps which is the computation and communication of FFT. We assume the errors are due to bit flips during the computation with a constant probability per instruction.

Computing Model: In our model, we assume that there are a total of S systematic processors, which have the original data and K coded processors, which store the encoded parity data. *Our computation goal is to simulate the interactions between N particles using $S + K$ processors, through computing 3D FFT of M^3 points (M: mesh size), while being resilient from bit flip errors during the computation.* For simplicity, we assume that M is divisible by S.

Communication Model: In our model, we assume the processors are fully connected and each processor can send or receive data at the same time (single duplex port model). We use an α-β model for communication cost. In this model, the time to send or receive a message of length l is given by:

$$T = \alpha + l \cdot \beta.$$

4. ALGORITHM DESCRIPTIONS

4.1. Parallel Molecular Dynamics Simulations

To evaluate the impact of coding on the performance of molecular dynamics simulations, we have designed an algorithm achieving a high level of parallelism as the underlying simulation platform. We partition the particles along one spatial dimension and each processor is responsible for calculating the force due to these particles. In our simulation, there are four stages where we realize parallelism, namely, van der Waals forces, short-range electrostatic forces calculations and Ewald mesh spreading via the spatial decomposition, as well as non-spatial decomposition based FFT part where our coding technique is applied. In the mesh Ewald method we compute the Coulomb interactions by first spreading the off-mesh particles to the grid points (discretization step), then using FFT to solve the Poisson equation in Fourier space and inverse FFT to send electrostatic potential back to real space, and finally transforming the on-mesh potentials to the forces on original particles. This leaves room for spatial decomposition based parallelization – each processor owns a part of the grid, and is responsible for the computations for the atoms in that part. The overall algorithm is described in Algorithm 1.

Algorithm 1: Parallel Molecular Dynamics Simulations

```
1  for each processor $p_x$ do
2      for each timestep $t_y$ do
3          for each particle $q_i$ in partition of $p_x$ do
4              for each particle $q_j$ within cutoff radius do
5                  calculate van der Waals interactions;
                   calculate short range electrostatic
                   interactions;
6          for each meshpoint $m_l$ do
7              for each charges $q_r$ within cutoff radius do
8                  add contribution of the charges to the
                   charge density $\rho_l$;
9          Perform 3d FFTs to get k-space densities $\tilde{\rho}_i$;
10         Solve Poisson equations on grid to get $\tilde{\phi}_i$;
11         Perform inverse 3d FFTs to get $\phi_i$;
12         for each particle $q_i$ in partition of $p_x$ do
13             calculate long range electrostatic
               interactions from mesh potentials;
14         Calculate bond forces;
15         Update velocity and position of particles;
```

4.2. Coded 3D FFT Algorithm

We will briefly explain the coded 3D FFT algorithm here. For more details, we refer to [7]. Our method differs from general distributed 3D FFT algorithms as we do not require the original arrangement of data in the final result for processing. Our method consists of three steps: 1) 2D FFT along the second and the third dimension, 2) transpose the first and the second dimension (all-to-all communication), 3) 1D FFT along the first dimension. We use optimal MDS codes for adding redundant computations before step 1 and 3.

5. COMMUNICATION COMPLEXITY ANALYSIS

Theorem 1. *Compared to uncoded FFT algorithm, Algorithm 2 has a communication overhead $T_{overhead}$ as follows:*

$$[6K\log(S)+2\log(\frac{S+K}{S})]\cdot\alpha+\frac{9KS-3K+2S\log(\frac{S+K}{S})}{S^2}M^3\cdot\beta. \tag{3}$$

Proof. The initial encoding step for the forward FFT involves K reduce(S,M^3/S) operations. This incurs communication [27] latency of:

$$T_{enc,fwd} = 2K\log(S)\cdot\alpha + 2KM^3/S\cdot\beta.$$

Similarly, initial encoding step for the inverse FFT involves K reduce(S,$2M^3/S$) operations, with the factor of 2 due to the representation of complex numbers. This has communication latency of:

$$T_{enc,inv} = 2K\log(S)\cdot\alpha + 4KM^3/S\cdot\beta.$$

Algorithm 2: Coded 3D FFT Algorithm

```
1  for each coded processor $p_i$ ($i = 1, \cdots, K$) do
2      Gather parity symbols from the $S$ systematic
       processors;
3  for each processor $p_i$ ($i = 1, \cdots, S+K$) do
4      Compute $M/S$ 2D FFTs of size $M^2$;
5      Encode parity symbols along second dimension;
6  All-to-all communication (transpose).
7  for each processor $p_i$ ($i = 1, \cdots, S+K$) do
8      Check for errors, and if there is an error, decode to
       recover the correct output;
9  for each processor $p_i$ ($i = 1, \cdots, S+K$) do
10     Compute $M^2/S$ 1D FFTs of size $M$;
11 for each coded processor $p_i$ ($i = 1, \cdots, K$) do
12     Scatter parity symbols to the $S$ systematic
       processors;
13 for each systematic processor $p_i$ ($i = 1, \cdots, S$) do
14     Check for errors, and if there is an error, decode to
       recover the correct output;
```

The All-to-all communication for both forward and inverse coded FFT involves an All-to-all($S+K$,$2M^3/S$) operation. This communication [28] costs:

$$T_{a2a} = \log(S+K)\cdot\alpha + \frac{M^3\log(S+K)}{S}\cdot\beta,$$

whereas for non coded FFT it costs:

$$T'_{a2a} = \log(S)\cdot\alpha + \frac{M^3\log(S)}{S}\cdot\beta.$$

The decoding for the forward FFT involves K reduce(S,$2M^3/S$) operations [29] whose communication cost is:

$$T_{dec,fwd} = K\log(S)\cdot\alpha + \frac{2K(S-1)M^3}{S^2}\cdot\beta.$$

Likewise, decoding operation for the inverse FFT involves K reduce(S,M^3/S) operations. This communication cost is:

$$T_{dec,inv} = K\log(S)\cdot\alpha + \frac{K(S-1)M^3}{S^2}\cdot\beta.$$

Therefore, total communication overhead of Algorithm 2 is

$$\begin{aligned} T_{overhead} &= T_{enc,fwd} + T_{enc,inv} + 2(T_{a2a} - T'_{a2a}) \\ &\quad + T_{dec,fwd} + T_{dec,inv} \\ &= [6K\log(S) + 2\log(\frac{S+K}{S})]\cdot\alpha \\ &\quad + \frac{9KS-3K+2S\log(\frac{S+K}{S})}{S^2}M^3\cdot\beta \end{aligned}$$

□

For molecular dynamics simulations, we are also concerned with the relation between overhead and the number of atoms. To provide consistent accuracy, the mesh size M must grow proportional to the number of atoms, N as follows:

$$M^3 \propto N.$$

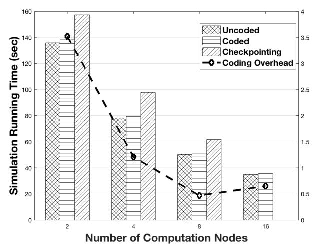
The overhead, $T_{overhead}$, can now be written in terms of N as follows:

$$[6K\log(S)+2\log(\frac{S+K}{S})]\cdot\alpha+\frac{9KS-3K+2S\log(\frac{S+K}{S})}{S^2}N\cdot\beta.$$

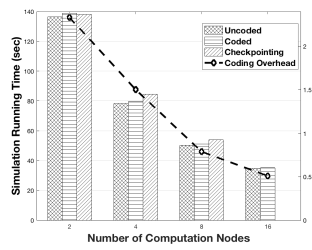
6. EXPERIMENTAL RESULTS AND DISCUSSION

We conducted our simulation on two proteins, a 22-atom alanine dipeptide (AD) in a 2270-atom system and a 255-atom folded short peptide (FSP) in a 11595-atom system. Their topology structures are shown in Figure 1. Our electrostatic force calculation is based on the FFP method from [23]. Following the original paper, we set the crucial parameters as follows: cutoff = 9Å, $\sigma = 3/\sqrt{2}$Å. Our program is implemented in Java with OpenJDK 1.8.0 using Open MPI 3.1.2 as the communication library and JTransforms 3.1 as the FFT library. All experiments are done in a single thread on Amazon EC2 M5.large instances with 2 vCPUs at 2.5GHz and 8 GiB of memory. To reduce the communication latency, all nodes are placed on the same cluster placement group. Experiments with no coded nodes and 2 coded nodes are performed with 2, 4, 8 and 16 systematic nodes on the two peptides for 100 time steps. Experiments with in-memory checkpointing (after each 10 steps) are also performed with up to 8 systematic nodes to serve as a benchmark for our algorithm[1].

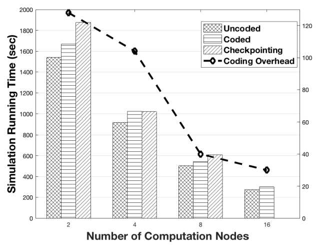
To demonstrate error resilience, we artificially injected errors between the FFT computation and All-to-all communication to simulate bit flips[2] during the computation of FFT. We used error probability $p = 1/128$ and $p = 1/256$, where p denotes the error probability per FFT step. We scaled p proportional to $1/S$ to make sure that the error probability per instruction does not change with varying S. Note that the computational steps per node is proportional to $1/S$. However, in our experiments, error probability per instruction is different in AD and FSP because we did not scale p with respect to the size of the computation, $M\log M$ or $M^2\log M^2$. This can be one possible explanation for the different behaviors in AD and FSP. In future experiments, we plan to fix the error probability over different protein molecules. Figure 2 shows the comparison of uncoded FFT, coded FFT, and checkpointing in terms of the total simulation time. The total simulation time includes both computation and communication. From the figure, we can observe that our coded FFT algorithm requires only slightly more simulation time compared to the uncoded implementation without error protection. Also, we noticed that even with coded FFT algorithm, our implementation roughly scales with the number of nodes, and furthermore the overhead of coding tends to decrease with the number of nodes. We attribute this to two reasons: 1) In the communication overhead given in Equation 3, β term dominates as S and K are small (at most 8), while M^3 can be very large (e.g., 32^3), 2) Up to 16 nodes, computation time might be more significant than communication time. In (1a) however, the overhead increases from 8 to 16 nodes, and we think this is due to the higher contribution from α term and relatively small M compared to FSP. Moreover, when the system is more prone to error, our coded FFT prevails over checkpointing due to more frequent restarts in checkpointing method under high error probability.



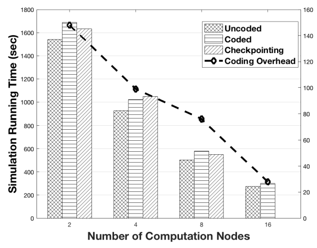
(1a) AD, $p = 1/128$



(1b) AD, $p = 1/256$



(2a) FSP, $p = 1/128$



(2b) FSP, $p = 1/256$

Fig. 2. Comparison results between uncoded FFT, coded FFT and traditional checkpointing technique under two typical error probabilities after 100 iterations. There is no error detection/correction for uncoded FFT.

7. LIMITATIONS AND FUTURE WORK

Our distributed FFT is currently decomposed along one dimension, which limits our scalability to M nodes. Incorporating 2D decomposition would allow scaling to M^2 nodes. Also, we can further optimize encoding and decoding communication suggested in [7]. Lastly, our results are limited to the soft-error case, especially, for bit-flip errors. There have been efforts to build ABFT support on MPI [19]. Incorporating our work with the MPI extension with fault tolerance to handle not only soft errors but also fail-stop errors or straggling processors would be an interesting future direction.

[1]This is a very optimistic estimate of the time overhead of checkpointing. Usually, checkpoints are stored in disks, and this incurs a significant time overhead.

[2]Note that our choice of error rate p is much higher than the error rate on today's supercomputers.

8. REFERENCES

[1] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2017.

[2] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, 2017.

[3] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding," *arXiv preprint arXiv:1612.03301*, 2016.

[4] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," *arXiv preprint arXiv:1705.03875*, 2017.

[5] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," *arXiv preprint arXiv:1705.10464*, 2017.

[6] Y. Yang, P. Grover, and S. Kar, "Coded distributed computing for inverse problems," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 709–719.

[7] H. Jeong, T. M. Low, and P. Grover, "Masterless coded computing: A fully-distributed coded fft algorithm," in *56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2018.

[8] F. Khalili-Araghi, J. Gumbart, P.-C. Wen, M. Sotomayor, E. Tajkhorshid, and K. Schulten, "Molecular dynamics simulations of membrane channels and transporters," *Current opinion in structural biology*, vol. 19, no. 2, pp. 128–137, 2009.

[9] J. L. Klepeis, K. Lindorff-Larsen, R. O. Dror, and D. E. Shaw, "Long-timescale molecular dynamics simulations of protein structure and function," *Current opinion in structural biology*, vol. 19, no. 2, pp. 120–127, 2009.

[10] M.-L. Bang, T. Centner, F. Fornoff, A. J. Geach, M. Gotthardt, M. McNabb, C. C. Witt, D. Labeit, C. C. Gregorio, H. Granzier *et al.*, "The complete gene sequence of titin, expression of an unusual 700-kda titin isoform, and its interaction with obscurin identify a novel z-line to i-band linking system," *Circulation research*, vol. 89, no. 11, pp. 1065–1072, 2001.

[11] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of computational physics*, vol. 117, no. 1, pp. 1–19, 1995.

[12] W. M. Brown, A. Kohlmeyer, S. J. Plimpton, and A. N. Tharrington, "Implementing molecular dynamics on hybrid high performance computers–particle–particle particle-mesh," *Computer Physics Communications*, vol. 183, no. 3, pp. 449–459, 2012.

[13] H. J. Berendsen, D. van der Spoel, and R. van Drunen, "Gromacs: a message-passing parallel molecular dynamics implementation," *Computer physics communications*, vol. 91, no. 1-3, pp. 43–56, 1995.

[14] W. M. Brown, P. Wang, S. J. Plimpton, and A. N. Tharrington, "Implementing molecular dynamics on hybrid high performance computers–short range forces," *Computer Physics Communications*, 2011.

[15] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward exascale resilience," *The International Journal of High Performance Computing Applications*, vol. 23, no. 4, pp. 374–388, 2009.

[16] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson *et al.*, "Addressing failures in exascale computing," *The International Journal of High Performance Computing Applications*, vol. 28, no. 2, pp. 129–173, 2014.

[17] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," in *Journal of Physics: Conference Series*, vol. 78, no. 1. IOP Publishing, 2007, p. 012022.

[18] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou, "Algorithm-based fault tolerance applied to high performance computing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 4, pp. 410–416, 2009.

[19] T. Naughton, C. Engelmann, G. Vallée, and S. Bohm, "Supporting the development of resilient message passing applications using simulation," in *2014 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2014, pp. 271–278.

[20] D. Hakkarinen and Z. Chen, "Algorithmic cholesky factorization fault recovery," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–10.

[21] J. Dongarra, T. Herault, and Y. Robert, "Fault tolerance techniques for high-performance computing," in *Fault-Tolerance Techniques for High-Performance Computing*. Springer, 2015, pp. 3–85.

[22] L. Tan, S. L. Song, P. Wu, Z. Chen, R. Ge, and D. J. Kerbyson, "Investigating the interplay between energy efficiency and resilience in high performance computing," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 786–796.

[23] D. York and W. Yang, "The fast fourier poisson method for calculating ewald sums," *The Journal of Chemical Physics*, vol. 101, no. 4, pp. 3298–3300, 1994.

[24] T. Darden, D. York, and L. Pedersen, "Particle mesh ewald: An n log (n) method for ewald sums in large systems," *The Journal of chemical physics*, vol. 98, no. 12, pp. 10 089–10 092, 1993.

[25] C. Sagui and T. Darden, "Multigrid methods for classical molecular dynamics simulations of biomolecules," *The Journal of Chemical Physics*, vol. 114, no. 15, pp. 6578–6591, 2001.

[26] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of computational physics*, vol. 73, no. 2, pp. 325–348, 1987.

[27] J. L. Träff and A. Ripke, "Optimal broadcast for fully connected processor-node networks," *Journal of Parallel and Distributed Computing*, vol. 68, no. 7, 2008.

[28] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems," *IEEE Transactions on parallel and distributed systems*, 1997.

[29] E. Chan, M. Heimlich, A. Purkayastha, and R. Van De Geijn, "Collective communication: theory, practice, and experience," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 13, 2007.