Parallelization of classical molecular dynamics with OpenMP

Samuel Moerman
Dept. of Computer Science
New York University
New York, USA
shm9853@nyu.edu

Anway Agte
Dept. of Computer Science
New York University
New York, USA
ama10345@nyu.edu

Atmaj Koppikar
Dept. of Computer Science
New York University
New York, USA
ark9238@nyu.edu

Mihir Prajapati
Dept. of Computer Science
New York University
New York, USA
mp6507@nyu.edu

***Abstract*—Molecular dynamics is a computational technique that provides atomic-level insights into material and chemical compound behavior by simulating interatomic interactions. This work presents a novel molecular dynamics engine with multicore parallelization strategies for electrostatic, dispersion, and bonded interactions.**

The proposed parallelization approach for electrostatic and dispersion interactions utilizes Newton's third law of motion to symmetrize interaction calculations, effectively reducing the computational complexity by eliminating redundant pairwise interactions. Extensive benchmarking reveals performance improvements that scale with increased thread count, highlighting the effectiveness of the proposed parallel decomposition strategy. In contrast, the non-bonded interaction parallelization exhibited limited scalability, failing to achieve linear performance improvements with increasing system size.

Additionally, a comprehensive analysis examines the cache locality and memory access patterns of the parallelization strategy, revealing insights into potential future thread-level performance optimization. Future work proposes exploring a hybrid CPU/GPU architecture using spatial decomposition techniques to further enhance performance and scalability.

***Index Terms*—Parallelization, OpenMP, Molecular Dynamics, Non-Bonded Interactions**

I. Introduction

Molecular dynamics (MD) is a computational technique that simulates the behavior of atoms and molecules by numerically solving their equations of motion. At its core, MD acts as a computational microscope, allowing researchers to observe and predict the dynamic behavior of molecular systems with at an atomic-level detail.

Molecular dynamics is based on the integration of Newton's equations of motion. For a system of N particles, this corresponds to the following equation:

$$m_i \frac{d^2\mathbf{r}_i}{dt^2} = \mathbf{f}_i = -\frac{\delta}{\delta \mathbf{r}_i} V(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) \tag{1}$$

Where $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N$ represent the set of nuclear coordinates of the N-particle system, $V(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$ the potential energy in function of these coordinates, $\mathbf{f}_i$ the force acting on particle i and m_i the mass of particle i.

The MD simulation follows a systematic computational workflow. An initial system configuration is provided to the program. Based on the initial positions of the particles in the system, the forces are calculated. These forces are then used to update each particle's position and velocity. These positions and velocities then form the input for the subsequent timestep and the cycle repeats itself. It is in this way that the program iteratively determines the behavior of the system, utilizing numerical integration techniques, such as the Verlet algorithm. This workflow is graphically represented in Fig. 1

Since this forms a system of non-linear differential equations not solvable analytically, MD relies on numerical integration algorithms to iteratively calculate particle positions and speeds throughout time. The power of MD lies in its ability to generate a comprehensive trajectory showing how the molecular system evolves, providing insights into atomic-level phenomena that are challenging to observe experimentally.

In this paper, the goal was to develop a comprehensive molecular dynamics (MD) simulation engine with a primary focus on parallel computational strategies. This encompassed the creation of a flexible and performant simulation framework capable of modeling interatomic interactions through the implementation of key physical force calculations, specifically targeting electrostatic, dispersion, and bonded interactions.

The core objectives of this work encompassed several important aspects of computational molecular dynamics and high-performance computing:

- Engine Development: Design a modular molecular dynamics engine that can simulate fundamental interatomic forces with computational efficiency and numerical precision. This involved creating a robust framework for particle representation, force calculation, and system evolution.
- Parallelization Strategies: Explore and implement advanced parallel computing techniques to overcome the computational complexity inherent in pairwise interaction calculations. By leveraging multicore processor architectures, the research aimed to dramatically reduce computational time for large-scale molecular simulations.
- Performance Optimization: Develop parallelization approaches that not only distribute computational load across multiple threads but also maximize cache utilization and minimize synchronization overhead. This required a deep understanding of both computational algorithms and hardware-level performance characteristics.

- Comparative Analysis: Systematically benchmark the parallelization strategies for various interaction types, critically evaluating their scalability, performance gains, and computational efficiency across different system sizes.

By focusing on these objectives, the research sought to contribute to the ongoing advancement in computational physics, demonstrating innovative approaches to parallelizing complex scientific computing tasks and providing a flexible framework for future molecular dynamics investigations.

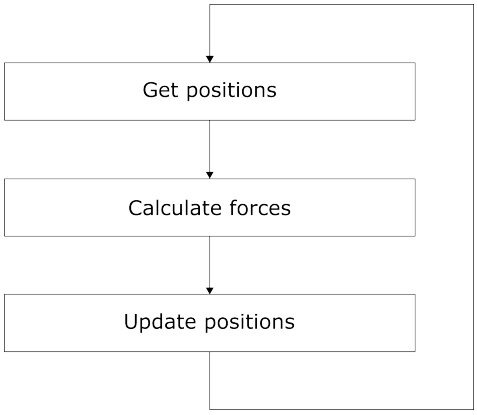

Fig. 1: Conceptual visualization of the core concept behind molecular dynamics

II. Molecular Dynamics Engine

Organic molecules and biological macromolecules are typically described with the following expression:

$$\begin{aligned} V(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) = V_{\text{bonds}} + V_{\text{angles}} + V_{\text{dihedrals}} \\ + V_{\text{impropers}} + V_{\text{dispersion interaction}} \\ + V_{\text{electrostatic interaction}} \end{aligned} \tag{2}$$

where $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N$ are the particles in the system, V is the potential energy of the system in function of these particles, V_{bonds} the potential related to the bonds, V_{angles} the potential related to the angles, $V_{\text{dihedrals}}$ the potential related to the dihedrals, $V_{\text{impropers}}$ the potential related to the improper torsions, $V_{\text{dispersion interaction}}$ the potential related to the dispersion interaction and $V_{\text{electrostatic interaction}}$ the potential related to the electrostatic interaction.

As described above, the formula consists of different types of potentials. The term V_{bonds} can be described as follows:

$$V_{\text{bonds}} = \sum_{\text{bonds}} \frac{k_d}{2} * (d - d_0)^2 \tag{3}$$

where d is the distance between two particles, d_0 the standard bond distance for that particular combination of particles and k_d the parameter associated with that specific bond. Not that this term is a sum of harmonic potentials. The energy increases as the bond distance deviates from the standard bond distance. When the bond distance reaches the standard bond distance, the energy associated with this bond becomes zero, which corresponds to a harmonic oscillator in a vacuum. The k_d variable is part of the set of parameters specific to the forcefield as discussed above.

The term V_{angles} can be described in a similar way:

$$V_{\text{angles}} = \sum_{\text{angles}} \frac{k_\theta}{2}(\theta - \theta_0)^2 \tag{4}$$

where θ represents the angle between a given set of 3 particles, θ_0 the standard angle for this particular set and k_θ the parameter associated with that specific angle. The interpretation of this term is similar to V_{bonds}, in that a deviation from the standard angle results in an increased energy.

A final term which is described by means of a harmonic oscillator is the term $V_{\text{impropers}}$, which is often introduced to maintain planarity of groups with a flat geometry or sometimes to preserve chirality:

$$V_{\text{impropers}} = \sum_{\text{impropers}} \frac{k_\psi}{2}(\psi - \psi_0)^2 \tag{5}$$

where ψ represents the angle between a set of 3 particles, ψ_0 the standard improper angle for that particular combination of particles and k_ψ the parameter associated with that specific improper angle. Again, the interpretation is similar to V_{bonds}, in that a deviation from the standard dihedral angle results in an increased energy.

The dihedral terms, also referred to as torsion terms, are responsible for introducing an energy barrier to the rotation of bonds which do not rotate freely. They can be expressed as follows:

$$V_{\text{dihedrals}} = \sum_{\text{dihedrals}} \frac{k_\phi}{2}(1 + \cos(n\phi - \phi_0)) \tag{6}$$

where ϕ represents the dihedral angle between a set of 4 particles, ϕ_0 the standard dihedral angle for that particular set of particles, k_ϕ the parameter associated with that particular dihedral angle and n the amount of minima in the rotational profile. The torsion terms are periodic in function of ϕ and are defined by two additional parameters, n and ϕ_0. Parameter n refers to the number of minima in the rotational profile and is also known as the multiplicity of the function. The parameter ϕ_0 is known as the phase factor, which determines the location of the minima. Often, the rotational profile of a dihedral function is not represented by one torsion function, but by a sum of multiple functions.

The dispersion interaction $V_{\text{dispersion interaction}}$ or van der Waals interaction is represented by the Lennard-Jones 126 potential and can be rewritten as follows:

$$V_{\text{dispersion interaction}} = \sum_{\substack{\text{non-bonded} \\ \text{pairs(i,j)}}} 4\varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] \tag{7}$$

where r_{ij} is the distance between a set of particles (i, j), ε_{ij} the maximum depth of the potential well for this set and σ_{ij} the distance between this set where the interaction is zero. This interaction takes into account both the attractive and the repulsive aspect of non-bonded interaction.

The electrostatic interaction $V_{\text{electrostatic interaction}}$ is represented by Coulomb's law and can be rewritten as follows:

$$V_{\text{electrostatic interaction}} = \sum_{\substack{\text{non-bonded} \\ \text{pairs(i,j)}}} \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}} \tag{8}$$

where r_{ij} is the distance between a set of particles (i, j), q_i and q_j are the charges on particles i and j respectively and ε_0 the electric constant. In reality, charges are dispersed throughout a molecule. Many forcefields, however, calculate electrostatic interaction by using Coulomb's law, assuming all charges are static and can be represented as point charges.

The Verlet integrator is a commonly used algorithm in molecular dynamics simulations to integrate Newton's equations of motion. It is favored for its numerical stability and time-reversibility properties. The positions of particles at time $t + \Delta t$ are calculated based on their positions at time t and $t - \Delta t$, along with the forces acting on them. The formula is given as follows:

$$\mathbf{r}_i(t + \Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t - \Delta t) + \Delta t^2 \cdot \frac{\mathbf{F}(t)}{m}, \tag{9}$$

where $\mathbf{r}(t)$ is the position of the particle at time t, Δt is the integration time step, $\mathbf{F}(t)$ is the force acting on the particle at time t, and m is the particle's mass.

This method explicitly depends on the current and previous positions of the particle, eliminating the need to directly compute velocities. However, velocities can still be calculated as a midpoint approximation:

$$\mathbf{v}(t) = \frac{\mathbf{r}(t + \Delta t) - \mathbf{r}(t - \Delta t)}{2\Delta t}. \tag{10}$$

Because the Verlet algorithm calculates positions and approximates velocities at a single time step, it allows for efficient computation without sacrificing accuracy in simulations of large molecular systems.

III. Literature Review

Molecular dynamics is a concept that is inherently very suited to parallelization. There are multiple MD engines that leverage parallelization, such as OpenMM. [3] Because the forces that drive the behavior of the system are comprised of the sum of interactions between sets of particles, a parallelization can be performed over these interactions. Moreover, because the technique is inherently an approximative technique, there is lots of freedom to further optimize the calculations by using smart approximative algorithms. In general, there are three main approaches that can be taken to improve the performance of the simulation engine.

A. Atom Decomposition

In this approach, each of the P processors are assigned a group of N/P atoms. These atoms do not necessarily need to have any spatial or other relation to each other. However, in order to update the positions of the atoms in a processor P_z, it would require the positions of many atoms that are not contained in this processor P_z. This requires each processor to receive updated positions of all other atoms that are interacting with the atoms in P_z, in an operation referred to as all-to-all communication. This all-to-all communication incurs a cost of $O(N)$. [4], [7]

The atom decomposition algorithms spread the force calculations and integrations evenly across the processors. However, because of this global communication, the algorithm does not scale well with the number of particles in the system. The main advantage of these algorithms is their simplicity. [4], [7]

B. Force Decomposition

In force decomposition approaches, each of the P processors are assigned to a block of particle pairs (instead of particles themselves). In other words, the skew-symmetric force matrix f that arises from Newton's third law of motion is spread out over the processors:

$$f = \begin{bmatrix} 0 & f_{12} & f_{13} & \cdots & f_{1N} \\ -f_{12} & 0 & f_{23} & \cdots & f_{2N} \\ -f_{13} & -f_{23} & 0 & \cdots & f_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -f_{1N} & -f_{2N} & -f_{3N} & \cdots & 0 \end{bmatrix}$$

Each processor is responsible for a sub-block of f of size $N/\sqrt{P} \times N/\sqrt{P}$. This is graphically illustrated in Fig 2

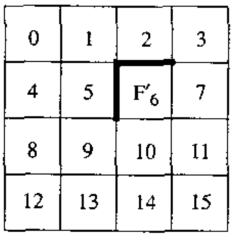

Fig. 2: Graphical representation of the force decomposition strategy. Modified from Plimpton. [7]

By permuting the force matrix f, the communication cost can be reduced from $O(N)$ to $O(N/\sqrt{P})$. Additionally, this method also does not require geometric information about the system being studies. In fact, to achieve better load balancing, it is better to randomize the atom ordering. [7]

C. Spatial Decomposition Methods

1) Domain Decomposition: Spatial decomposition methods are a fundamental approach to parallelizing MD simulations. The central idea is to split the simulation region into boxes that are then independently processed. The volume of these boxes is chosen such that the load between the cores is optimally balanced. More modern approaches often tackle this issue in a dynamic fashion, by readjusting the boxes as the simulation progresses.

Each of the processors is responsible for simulating the behavior of the particles in that region. One of the main challenges of this approach is how the interfaces between the subdivisions are handled. Many approaches leverage the fast multipole algorithm propsed by Greengard and Rothkin [6].

Other approaches could include the Barnes-Hut algorithm [2]. These spatial decomposition techniques are highly suited to CPU parallelization because of the irregular branching, adaptive recursion and non-uniform memory access patterns.

2) Neighbor Lists: Neighbor list is another spatial decomposition approach that can be used to parallelize MD simulations. The core idea is to maintain a list of potential interaction partners for each particle in the system. This significantly reduces the time complexity, given that the algorithm no longer needs to iterate over all atom pairs ($O(N)$), but only needs to iterate over the neighbor list for each particle ($O(N)$). These techniques are especially relevant for short-range non-bonded interactions. This technique is especially suited to CPU, given that it is extremely challenging to coalesce the memory, which is required for efficient GPU parallelization. [1], [8]

IV. Parallelization

The proposed parallelization approach sets out to parallelize the force calculations for the involved particle pairs. A directed acyclic graph of the process that is being parallelized is represented in Fig. 3

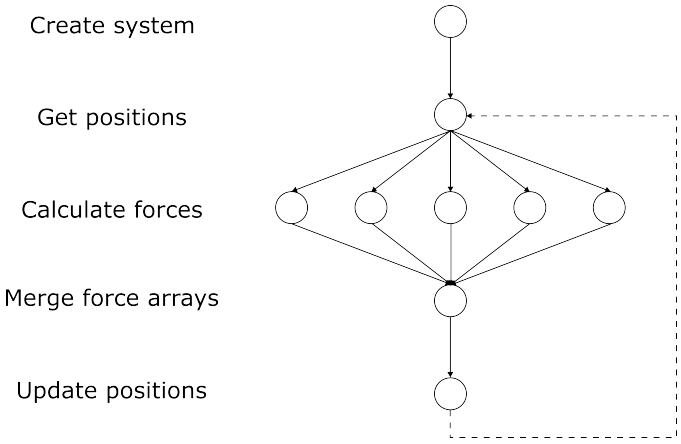

Fig. 3: A direct acyclic graph illustrating the computations performed in the parallelized section of the code. Note that the dashed line illustrates that this DAG is executed for multiple timesteps to simulate the behavior of the system.

A. Electrostatic Forces

In general, the non-bonded interactions (i.e. the electrostatic interaction and the dispersion interaction) usually take up the majority of the time for classical molecular dynamics simulations. This is because these interactions are two-particle interactions, which would naively require a double loop over all particles resulting in an $O(N^2)$ time complexity.

```
compute(system):
    for particles in system:
        for particles in system:
            - calculate distance
            - compute electrostatic
              force
```

However, because of Newton's third law of motion, it is possible to leverage the concept of a skew-symmetric force matrix f

$$f = \begin{bmatrix} 0 & f_{12} & f_{13} & \cdots & f_{1N} \\ -f_{12} & 0 & f_{23} & \cdots & f_{2N} \\ -f_{13} & -f_{23} & 0 & \cdots & f_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -f_{1N} & -f_{2N} & -f_{3N} & \cdots & 0 \end{bmatrix}$$

of which only the upper or lower triangular portion needs to be calculated.

```
compute(system):
    for particle pairs in system:
            - calculate distance
            - compute electrostatic force
```

This loop is parallelized using a `collapse(2)` clause, which can deal with non-rectangular loops since OpenMP 5.1.

```
compute(system):
    #pragma omp for collapse(2)
    for particle pairs in system:
            - calculate distance
            - compute electrostatic force
            - store in local force vector

    #pragma omp critical
    - merge local force arrays
```

Note that thread-local force vectors are used during the calculations.

Because this loop is collapsed, the work across the iterations is uniform. Therefore, no schedule was needed to obtain optimal performance (which is not possible either way with a collapsed non-rectangular loop). Alternative setups with a schedule were also attempted, but the collapsed loop gave the best performance.

B. Dispersion Forces

This force type follows the same scheme as the electrostatic forces.

```
compute(system):
    #pragma omp for collapse(2)
    for particle pairs in system:
            - calculate distance
            - compute dispersion force
            - store in local force vector

    #pragma omp critical
    - merge local force vectors
```

C. Bonded Forces

The bonded forces represent the bonds between two particles. They are created at the system creation stage, and are stored in an `std::vector`. Besides this, the local force array concept is maintained from the non-bonded forces.

```
compute(system):
    #pragma omp for
    for bonds:
        - calculate distance between
            particles in bond
        - compute dispersion force
        - store in local force vector

    #pragma omp critical
    - merge local force vectors
```

Again, because of the uniform work across all the iterations of the loops, introducing a schedule did not result in an increased performance due to the additional overhead.

V. Results

In this section, the results of the parallelization strategies will be reported. The benchmarking was performed on the Crunchy 1 server, which contains a Four AMD Opteron 6272 (2.1 GHz) processor with 64 cores. Additionally, 256 GB of memory is available. All measurements were performed in triplicate and averaged out. In order to make a fair assessment of the parallelization, this speedup is calculated relative to a proper sequential version of this code rather than a parallel version run with a single thread. This way, overhead associated with ensuring thread-safe memory access is also eliminated and the true performance of parallelization can be accurately assessed.

In the following subsections, plots of the speedup in regard to the number of threads will be given. These results will then be discussed in Section VI Note that the x-axis of these plots is in a logarithmic scale, to represent the speedup over the full range of threads in a clear manner.

A. Electrostatic Interaction

In Fig. 4, a plot of the speedup of the electrostatic force calculations in regard to the number of threads is displayed for a range of different system sizes.

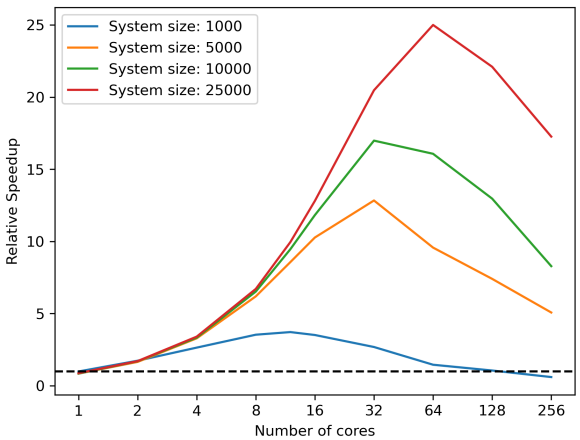

Fig. 4: Speedup of a range of systems containing only electrostatic interaction in function of the number of threads

A clear speedup can be observed for all system sizes. For the smallest system sizes, a speedup is achieved by leveraging multiple cores. However, this behavior only holds to a certain number of cores, after which there is a decrease in the speedup to the point where the sequential version becomes more performant. When increasing the system size, the parallelization is more scalable. In other words, more cores can be leveraged to achieve an even greater speedup. For the largest system size of 25000, the maximum speedup corresponds to the 64 logical cores that are available in the testing system. However, increasing the number of cores beyond this number also decreases performance.

Additionally, some manual benchmarking was done to calculate the time spent per individual interaction. This seemed to be quite independent of the system size, and resulted in a maximum time spent on an interaction around $400ns$.

B. Dispersion Interaction

In Fig. 5, a plot of the speedup of the electrostatic force calculations in regard to the number of threads is displayed for a range of different system sizes.

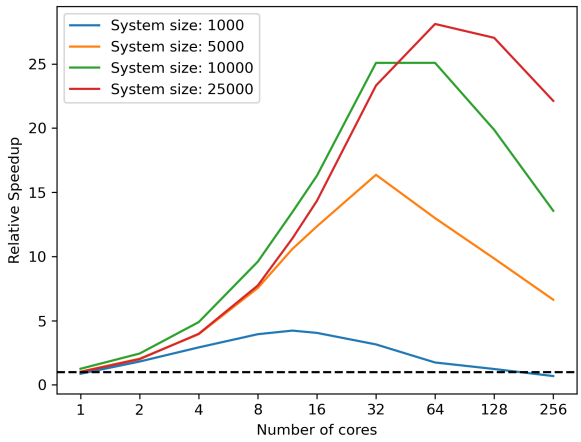

Fig. 5: Speedup of a range of systems containing only dispersion interaction in function of the number of threads.

The trends for the dispersion interaction are largely similar to the electrostatic interactions, given that they iterate over the same interaction pairs. However, the maximum obtained speedup seems to be slightly larger than the electrostatic interactions.

Additionally, some manual benchmarking was done to calculate the time spent per individual interaction. This seemed to be quite independent of the system size, and resulted in a maximum time spent on an interaction around $400ns$. Note that despite the increased complexity of this calculation, the time is more approximately equal to that of an electrostatic interaction.

1) Bonded Interaction: In Fig. 6, a plot of the speedup of the electrostatic force calculations in regard to the number of threads is displayed for a range of different system sizes. In

Fig 7, a similar benchmark is displayed, but for larger system sizes and smaller number of threads.

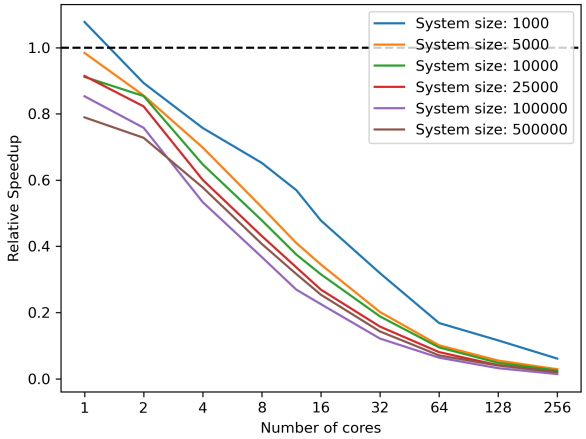

Fig. 6: Speedup of a range of systems containing only bonded interaction in function of the number of threads.

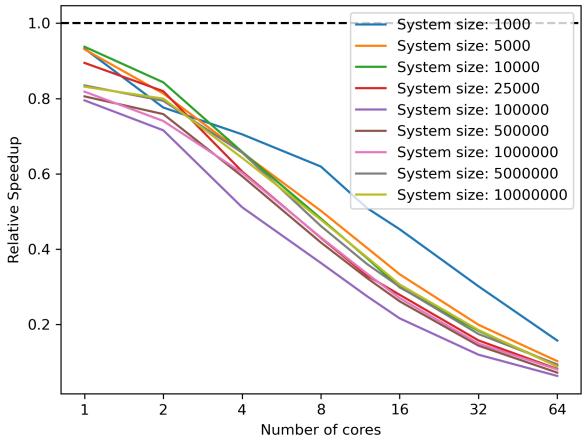

Fig. 7: Speedup of a range of systems containing only bonded interaction in function of the number of threads.

Here we see no significant performance increase for the parallelization of the bonded forces. There is no clear trend over the system sizes, but we can note that the smaller system sizes seem to drop in performance slightly less steeply than the larger system sizes.

VI. Discussion

A. Non-bonded Interactions

Leveraging the skew-symmetric force matrix, it is possible to eliminate a large section of the particle pairs that normally would be iterated over. However, this results in a non-rectangular loop, which normally would not result in a uniform load balance. However, because of the `collapse(2)` cause, a uniform load balance is achieved after all. This was also clearly observed during the development stage, where the timings for all threads were very uniform.

We can clearly see the best speedup and scaling for non-bonded interactions (i.e. the electrostatic forces and the dispersion forces). This is because of the collapsed double loop, which drastically increases the amount of work per thread. As illustrated in Fig 5 and Fig 4, small system sizes only experience a speedup up to a limited number of cores. This is because a balance has to be found between the overhead associated with managing multiple threads and the performance gain. As the number of cores is increased, the overhead increases to the point where there is no performance gain with regard to the sequential code. As the system size is increased, the amount of work that can be split over the threads also increases. Subsequently, the overhead associated with managing these threads is also spread out more. This is the reason why the larger system sizes experience performance gains for a larger number of threads. Note that the number of threads that correspond to the largest performance gain corresponds to the number of logical cores in the system. Beyond this point, the parallelism does not really increase anymore, because only concurrency is achieved.

As stated in Section IV, the non-bonded force calculations make use of local force vectors to store the calculated forces per thread. Subsequently, a critical section is used to merge these local force vectors into the global vector. This was done for correctness and optimization purposes. It is entirely possible that two threads are working on two separate interactions that happen to involve a common particle (e.g. forces f_{ij} and f_{ik} both involving particle i). If these two were to write to the global force vector, this would result in a race condition and an incorrect end result. In these physical simulations, that is of course unacceptable. To mitigate this, a locking data structure could potentially be used. Alternatively, a critical section that merges these thread-local forces back into the global force array in a thread-safe manner could also be used. During development, it turned out that the latter option was more performant. Another alternative could be a reduction clause on these force arrays, but attempts to implement this were not successful (even though it may be even more performant than the approach with the critical section).

While Amdahl's law is a theoretical concept, it could potentially help to give further insight into the behavior of the program. More specifically, it could give some theoretical insight to how much of the runtime is spent on performing parallel calculations and how much is spent on the sequential part and the overhead. As stated in Section V, the maximum time measured for a single interaction was around $400ns$ for both the electrostatic and dispersion interaction. Some calculations were made to try to estimate how much of the

runtime of the program was spent on the parallel section:

$$\text{iterations} = \frac{n * (n-1)}{2}$$

$$\text{time spend on parallel region} = \frac{\text{iterations} * 400ns}{\text{num cores}}$$

However, it became clear that the resolution of the internal clock was much lower than what was required to accurately time the calculations for one interaction. Future work could bundle these interactions and average over the number per batch to get a better estimate. However, for this work, no theoretical estimate could be made on the ratio of time spent in the parallel to the sequential and overhead region.

Upon system creation, the atoms are stored in a vector. As a result, the atoms that are contiguous in index are also contiguous in memory, which could provide some cache locality benefits during the execution. More specifically, it is possible that when bringing atom with index i into memory, the atom with index $i+1$ is also brought into memory. Given that this atom with index $i+1$ is next in line for calculation, it could provide an optimal cache access. However, if the atom with index $i+1$ is given to another thread, this cache locality will not be exploited. OpenMP currently does not support a scheduling strategy with the collapsed non-rectangular loops. Therefore, the scheduling is transparent to the programmer and could potentially result in a sub-optimal memory access pattern. If the scheduling uses some sort of chunking, where a sequence of contiguous atoms are processes by the same processor, this locality would be exploited.

Even more cache locality could be exploited if the data structure was changed from a vector of structs to a struct of vectors. This is because that way, the cache locality is even more likely to improve due to better alignment of the data structures. More specifically, each vector would store a specific property of the atoms in a contiguous vector. These vectors (e.g. the vector containing the positions of the atoms) would contain the data of a larger number of atoms, which would reduce the number of cache misses.

B. Bonded Interactions

In contract to the non-bonded forces, the bonded forces are not subject to the double loop over the particles. The load balance is uniform because the loop that is being parallelized performs the same calculation over all bonds present in the system. However, no performance speedup is observed at all. The amount of work per thread is just not enough to justify the overhead associated with managing the threads.

Additionally, the resolution of the clock of the computer is again not fine enough to get insight into the parallelization according to Amdahl's law.

It is important to note that all of these interactions are SIMD calculations. So while they are very effectively scaled up by means of a CPU, it is very likely that a GPU would be able to accomplish even better performance (given its inherent aptitude to SIMD instructions). In the field of molecular modeling, it has frequently been shown that GPU processing gives the best performance. [3], [5]

VII. Conclusion

A molecular dynamics simulation engine is a clear ideal candidate for parallelization. A clearly scalable parallelization strategy was implemented for the calculations of non-bonded interactions of moderate to large system sizes. From the data that was presented, it seems plausible that increasing the number of cores in the system would allow the strategy to scale even further beyond the systems sizes that were studied in this work.

The bonded interactions were not successfully parallelized with the proposed strategy. It seems plausible that increasing the system size would result in a better parallelization, but the data does not give the same indication. It could be the case that the overhead incurred by parallelizing on a CPU does not weigh up to the performance gain on systems sizes that are currently feasible to simulate.

Further minor improvements could potentially be made by adjusting the datastructures to allow for greater cache locality and implementing reduction clauses to potentially improve performance. More fundamental improvements are likely to be achievable by implementing more advanced parallelization strategies, such as those presented in the literature review. These would include atom-based decomposition techniques, force-based decomposition techniques and finally spatial decomposition techniques. Of this spatial decomposition technique category, an adaptive fast multipole method seems very likely to result in drastic performance increases with CPU parallelization (vide supra). Even more ambitious increases may be achievable for both non-bonded interactions and bonded interactions by using GPU to more efficiently parallelize these SIMD processes. Notably, a hybrid GPU/CPU nested parallelism architecture could be leveraged to employ a spatial decomposition technique such as the fast multipole method. This technique could be parallelized on CPU, and then delegate the computations in the resulting subregions to a GPU-based parallelization approach similar to the approach proposed in this work.

References

[1] Michael P Allen and Dominic J Tildesley. *Computer simulation of liquids*. Oxford university press, 2017.

[2] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986.

[3] Peter Eastman and Vijay Pande. Openmm: A hardware-independent framework for molecular simulations. *Computing in science & engineering*, 12(4):34–39, 2010.

[4] GC Fox, M Johnson, G Lyzenga, S Otto, J Salmon, D Walker, and Richard L White. Solving problems on concurrent processors vol. 1: General techniques and regular problems. *Computers in Physics*, 3(1):83–84, 1989.

[5] Mark S Friedrichs et al. Accelerating molecular dynamic simulation on graphics processing units. *Journal of Computational Chemistry*, 30(6):864–872, 2009.

[6] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

[7] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995.

[8] Loup Verlet. Computer" experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical review*, 159(1):98, 1967.