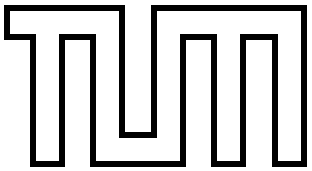


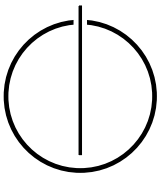
FAKULTÄT FÜR INFORMATIK

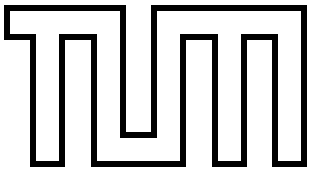
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Exploring Adaptive Resolution Simulation for Large Scale Systems with ls1-Mardyn

Alex Hocks





FAKULTÄT FÜR INFORMATIK

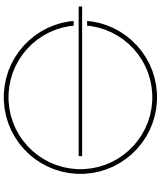
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Exploring Adaptive Resolution Simulation for Large Scale Systems with ls1-Mardyn

Evaluation von Simulationen mit Adaptiver Auflösung bei Großsystemen mit ls1-Mardyn

Author:	Alex Hocks
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Samuel Newcome, M.Sc.
	Amartya Das Sharma, M.Sc.
	Prof. Dr. rer. nat. habil. Philipp Neumann
Date:	16.08.2023



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 16.08.2023 Alex Hocks

Acknowledgements

During my months of working on this thesis, I encountered some challenges, all of which were resolved through the help of many people. I want to thank them sincerely for their support.
First and foremost, I appreciate the opportunity Prof. Dr. Hans-Joachim Bungartz and Prof. Dr. Philipp Neumann made possible for me to work on the joint project of the chair of Scientific Computing and Computational Science at the Technical University of Munich and the chair of High Performance Computing at the Helmut-Schmidt-Univerity in Hamburg.
I am grateful to my advisors, Samuel Newcome and Amartya Das Sharma, who provided insight and valuable hints whenever I was uncertain and spent many hours in the weekly meetings with me over the course of these months.
Most of my work would not have been possible without access to the supercomputers. Therefore, I am glad to have been provided with the facilities at HSU and LRZ and the help of their support team.
Finally, I thank my friends and family for their unwavering support, whether directly through discussions or indirectly through giving me the ideal working environment.

Abstract

Molecular Dynamics (MD) simulations approximate the movement over time of all atoms in a system of molecules. The computational effort of common approaches is proportional to the number of molecules. As the systems of interest grow in size or in simulation duration, computing resources become a limitation and must be used efficiently. The adaptive resolution scheme (AdResS) is a technique to reduce computational complexity by allowing the dynamic change of resolution during a simulation. In this thesis, we investigate a version of AdResS, in which molecular representations are gained by experimental measurements and not by RDF sampling. First, we discuss different weight functions. Then, we implement this AdResS variation in an existing MD-Platform ls1-Mardyn and perform different benchmarks to evaluate its accuracy and performance. We conclude that this method can achieve acceptable accuracy whilst providing a speedup when used with appropriate parameters.

Zusammenfassung

Molekulardynamik (MD) Simulationen stellen approximative Berechnungen der Bewegung aller Atome innerhalb eines Systems an Molekülen im Verlauf der Zeit bereit. Der Berechnungsaufwand gängiger Ansätze ist proportional zu der Anzahl an Molekülen. Mit der wachsenden Größe und Simulationsdauer der gewünschten Systeme werden Rechenressourcen mehr beansprucht und müssen somit effizient verwendet werden. Das adaptive Auflösungs Schema (engl. AdResS) ist eine Methode, um die Berechnungskomplexität zu verringern, indem das dynamische Ändern der Auflösung während der Simulation ermöglicht wird. In dieser Arbeit untersuchen wir eine Variation von AdResS, in welcher die molekularen Darstellungen durch experimentelle Messungen und nicht durch RDF Daten erzeugt werden. Des weiteren implementieren wir AdResS in einer existierenden MD-Platform, ls1-Mardyn, und führen unterschiedliche Tests durch zur Untersuchung dessen Genauigkeit und Leistung. Es stellt sich heraus, dass diese Methode befriedigende Genauigkeit in Zusammenhang mit einer Erhöhung der Leistung bereitstellen kann, wenn angemessene Parameter gewählt werden.

Contents

Acknowledgements vii

Abstract ix

Zusammenfassung xi

I. Introduction and Background 1

1. Introduction 2

2. Molecular Dynamics Basics 3

3. Introduction to AdResS 5
3.1. Computational Model 5
3.2. Coupling of Full Particle and Coarse Grain regions 7
3.3. Weight Function 8

4. Related Work 12
4.1. Grand Canonical AdResS (GC-AdResS) 12
4.2. Open Boundary MD 12
4.3. Hamiltonian AdResS 12

II. Implementation in ls1-Mardyn 13

5. ls1-Mardyn Architecture 14
5.1. Simulation 14
5.2. Domain Decomposition 15
5.3. Integrator 15
5.4. Particle Container 15
5.5. Cell Processor 16
5.6. Pair Handler 16

6. Particle Container Design 17
6.1. Approach: 1 Particle Container 17
6.2. Approach: 3 Particle Container 17
6.3. Evaluation 18

7. AdResS Implementation **19**
7.1. Data Representation 19
7.2. Plugin 20
7.3. Force Computation 21

8. Accuracy and Performance **22**
8.1. Benchmark Scenario 22
8.2. Accuracy 25
8.3. Performance 34
8.3.1. Results on HSUper 34
8.3.2. Results on CoolMuc 2 41
8.3.3. Conclusion of results 43

9. Summary and Conclusion **44**

III. Appendix **45**

A. Using AdResS in ls1-Mardyn **46**
A.1. Acquiring ls1-Mardyn 46
A.2. Input Files 46
A.3. Compiling ls1-Mardyn 47
A.4. Manually Running Scenarios 47

Bibliography **50**

Part I.

Introduction and Background

1. Introduction

Molecular Dynamics (MD) simulations aim to approximate the movement over time of all atoms in a system of molecules. By adhering to the laws of physics, natural behaviour should be recreated, which presents an opportunity to gain information about the macroscopic structure and interactions of molecules. This is useful for understanding bio-molecules or the folding of proteins and is often accompanied by experiments to verify data. [6, 1] For specific scenarios, no experiments can be conducted or are not viable, such as creating and observing a mixture of supercritical fuel and oxygen. [7]

When performing any MD simulation, all molecules interact with each other. Therefore, forces must be computed between all atoms of any given molecule interaction pair. Without further optimisation, this leads to a quadratic growth of computations. Considering that, firstly, simulations of, for example, the solvation of proteins in water can contain multiple millions of molecules [3] and, secondly, simulations of long time scales are desired, the resulting needed computational load is often very high. An example simulation of 10 million molecules and 100,000 time-steps would take approximately one month on an Intel® i7 12700KF @ 5.0GHz. Therefore, these simulations are usually run on supercomputers, which provide state-of-the-art facilities, such as fast memory on the order of terabytes, processors of high core counts or optimal topologies with fast interconnect technologies. Regardless, there are certain limitations in terms of simulation size or length. In some cases, the computational effort can be reduced by compromising accuracy, such that the number of interactions is linear in the number of molecules. However, that is not feasible for other scenarios, as accuracy is so important that quantum mechanics is used to describe the system. But because of this demand for accuracy, the workload is significantly higher. [2]

The Adaptive Resolution Scheme (AdResS) proposes a solution or compromise for such and similar situations. AdResS allows for the dynamic change of resolution during a simulation. Here, and in the rest of the thesis, **resolution** refers to the level of detail of a molecule. A high-resolution representation is the most faithful to the original molecule's structure and a low-resolution representation simplifies the molecule, often to a single point in space. But care has to be taken to not lose important details in the simpler version of the particle. By doing so, simulations of even greater lengths or higher complexity are achievable. [10]

This thesis aims to explore an altered version of AdResS that does not enforce single particle representations for the low-resolution molecules. Experiments in this thesis test if this alteration impacts the level of accuracy. Additionally, we explore the implementation in an existing large-scale MD-Platform, ls1-Mardyn, regarding implementational effort and potential considerations. Finally, we analyse the performance of our implementation on different supercomputers.

First, we discuss necessary background information about Molecular Dynamics and AdResS in chapters 2 and 3. At the end of the introduction to AdResS, we also present different weight function approaches. Afterwards, we briefly highlight other AdResS variations in chapter 4. In the central part of the thesis, we first describe the architecture of our MD-Platform, ls1-Mardyn, in chapter 5. Based on the architecture, our considered design and accompanying decisions during the implementation process are highlighted in chapter 6. Then the actual implementation is explained in chapter 7. As the last topic, we evaluate the accuracy and performance of our AdResS implementation on different supercomputers, see chapter 8.

2. Molecular Dynamics Basics

As molecular dynamics is classified by an N-body problem, a formal definition is given: There exist N particles, with each having a position r, velocity v and mass m. For a particle $i \in 1, ..., N$ those are denoted by: r_i, v_i and m_i. Let t_0 be the starting time. The problem's objective is to find r_i and v_i for all bodies i across all time steps. This is done by computing the interactions between all particles. The movement is governed by Newton's equations of motion:

$$\frac{dr_i}{dt} = v_i \tag{2.1}$$

$$\frac{dv_i}{dt} = a_i \tag{2.2}$$

with a_i being the acceleration of particle i. Due to the nature of these equations, an approximating integration step is necessary, as we only consider discrete time steps. Following Newton's second law, the acceleration is acquired by:

$$a_i = \frac{F_i}{m_i} \tag{2.3}$$

F_i is the acting force on particle i, which is composed of the contributions of the interactions with all other particles $j \neq i$:

$$F_i = \sum_{j=1, j \neq i}^{N} F_{ij} \tag{2.4}$$

F_{ij} depends on the application. As we consider molecular dynamics simulations, it is defined as the negative derivative of a potential function U in regard to the distance r_{ij} between particles i and j:

$$F_{ij} = -\frac{dU}{dr_{ij}} \tag{2.5}$$

The potential function describes the strength and kind of interaction between two particles. Further detail are given in section 3.1. [16]

A basic implementation for the main simulation loop, which adheres to the stated equations for physical accuracy, requires the following steps:

1. Update the position of all molecules.
2. Compute the interactions between all molecules, i.e. for each molecule, calculate the effect each other molecule has on it.
3. Update the velocity of all molecules.

With no further optimizations, the complexity of the force computation is within $\mathcal{O}(N^2)$. One common optimization strategy is to introduce a cutoff radius r_c. In that case, forces for particles that are farther apart than r_c are no longer computed. Examples are the Linked Cells and Verlet Lists methods. In Linked Cells, the simulation domain is split into cells of size r_c. During force computation, only the forces between particles within r_c with respect to the currently handled particle i are computed. This leads to a complexity of $\mathcal{O}(N)$. In Verlet Lists, every particle manages a list of neighbours within a "skin thickness" r_s. This approach requires fewer comparisons for whether the distance is less than r_c. In exchange, it uses more memory for potentially higher performance. [5, 13, 16]

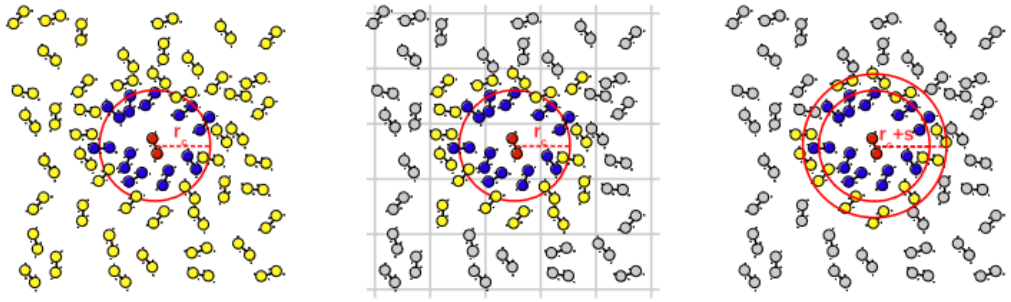


Figure 2.1.: "Methods for short-range force calculation for the molecule in red. The molecules in blue are the interaction neighbours of the molecule in red. The distance to molecules in yellow is checked by the respective algorithms to determine whether they are neighbours of the molecule in blue or not. Distances to molecules in grey do not need to be checked by the respective algorithms. (a) Direct N 2 calculation. (b) Linked Cells method. (c) Verlet Lists method" [16]
Source: Tchipev 2020 [16]

3. Introduction to AdResS

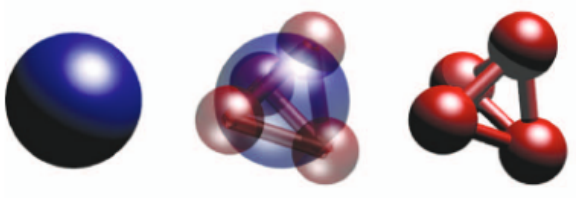


Figure 3.1.: Molecular representation of a hypothetical tetrahedral molecule.
CG: left H: middle FP: right
Source: Praprotnik et al. 2005 [10]

AdResS is a method which allows the change of the resolution of molecules during the simulation to reduce computational effort and thus increase performance. Fundamentally, it uses two representations for one kind of molecule. The first reflects the molecule in full or higher detail. It can contain all atoms but may also use functional groups or similar. Therefore, we call it Full Particle (FP) in the remainder of the thesis. Its counterpart has a lower resolution and can just be represented by a ball in the coarsest case. We call this representation Coarse Grain (CG). An example is given by an imaginary tetrahedral molecule, which has first an FP representation and then is simplified into a single CG sphere, in figure 3.1. The overlay of the molecules in the middle of the figure is explained in section 3.2.

3.1. Computational Model

As defined in equation 2.5, forces are computed through a potential function. This means that we do not model molecules as a spatial arrangement of atoms but rather represent them by tuning the parameters of potential functions. The potentials we use are of two kinds. The first is the Lennard-Jones (LJ) potential. It represents the mass-carrying component and thus acts in short ranges repulsively and in long ranges as Van-der-Waals forces. The other kind falls into the electrostatics category and models different charges. Using these potentials we construct molecules as an arrangement of "sites". Each site can exert either an LJ or charge potential.

For Full Particle resolution, no simplification is performed. Therefore, all molecules are represented by all of their atoms. These are represented through the use of multiple sites. The form of Lennard-Jones potential we use is the 12-6 Lennard Jones potential defined by:

$$U_{ij}^{LJ} = 4\epsilon_{ij}\left(\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right) \tag{3.1}$$

r_{ij} denotes the distance of site i to site j. ϵ and σ describe the properties of the underlying atom or molecule. Thus, sites of the same molecules have equal values for ϵ and σ. In order to interact sites of different kinds, these values are mixed using the following rules:

$$\epsilon_{ij} = \sqrt{\epsilon_i \cdot \epsilon_j} \tag{3.2}$$

$$\sigma_{ij} = \frac{\sigma_i + \sigma_j}{2} \tag{3.3}$$

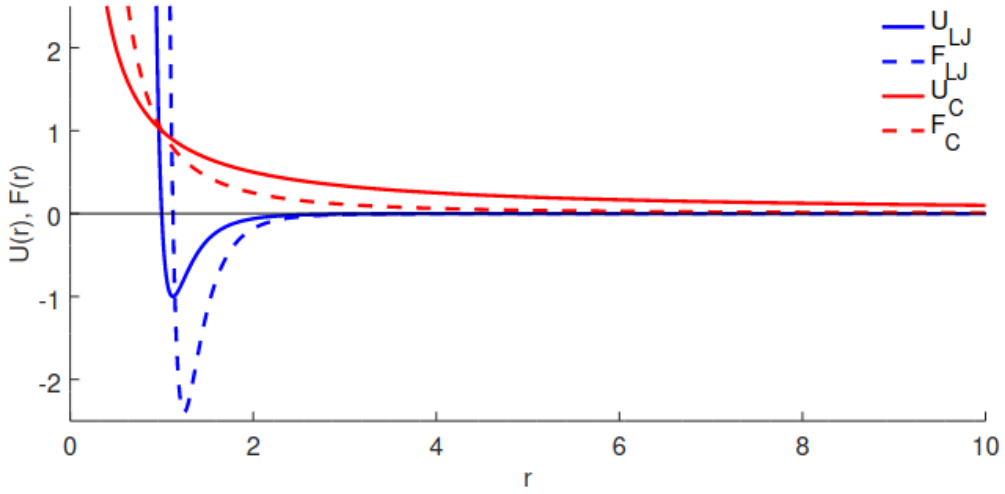


Figure 3.2.: Potential and resulting force functions for Lennard-Jones and Coulomb potential.
Source: Source: Tchipev 2020 [16]

The electrostatic potential models charges through the Coulomb potential. When viewing two point charges i and j, their potential is given by:

$$U^{qq}(r_{ij}) = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}} \tag{3.4}$$

q_i and q_j are the charge amounts of their respective sites. r_{ij} again denotes the distance and ϵ_0 represents the vacuum permittivity constant. When charges are placed close to each other, their interaction with other charges can be simplified. Two charges of different polarity but of the same magnitude can be merged into a dipole. Analogously two dipoles can be merged into a quadrupole. The dipole-charge $U^{q\mu}$ and dipole-dipole $U^{\mu\mu}$ interaction are given by:

$$U^{q\mu}(r_{ij}, \omega_j) = \frac{1}{4\pi\epsilon_0}\frac{q_i \mu_j}{r_{ij}^2} \cdot f_1(\omega_j) \tag{3.5}$$

$$U^{\mu\mu}(r_{ij}, \omega_i, \omega_j) = \frac{1}{4\pi\epsilon_0}\frac{\mu_i \mu_j}{r_{ij}^3} \cdot f_2(\omega_i, \omega_j) \tag{3.6}$$

r_{ij} is the distance, μ_j the dipoles charge and f_1, f_2 dimensionless functions of the dipoles

orientation angle ω. Following the same pattern, the interactions with quadrupoles Q are:

$$U^{qQ}(r_{ij}, \omega_j) = \frac{1}{4\pi\epsilon_0} \frac{q_i Q_j}{r_{ij}^3} \cdot f_3(\omega_j) \tag{3.7}$$

$$U^{\mu Q}(r_{ij}, \omega_i, \omega_j) = \frac{1}{4\pi\epsilon_0} \frac{\mu_i Q_j}{r_{ij}^4} \cdot f_4(\omega_i, \omega_j) \tag{3.8}$$

$$U^{QQ}(r_{ij}, \omega_i, \omega_j) = \frac{1}{4\pi\epsilon_0} \frac{Q_i Q_j}{r_{ij}^5} \cdot f_5(\omega_i, \omega_j) \tag{3.9}$$

with Q being the quadrupole charge. Molecules have their central position. Relative to that, sites are placed at a fixed offset. Therefore, no intra-molecular forces are computed.

With the full-resolution model defined, we need to find a model for the Coarse Grained representation. First, we present the method commonly used in literature to highlight the difference in our approach.

In the papers by Praprotnik et al. [10, 11, 12], the Full Particle model is mapped to a Coarse Grained one by aggregating all sites into a single one, with the mass being the sum of all previous sites. Thereby, in their work they reduce the number of rotational DOF and site-site interactions. A new pair potential for this Coarse Grain model is created in the next step. This is done by generating the full-resolution model's centre-of-mass radial distribution function $\mathrm{RDF}_{c.m.}$. Then, the potential is obtained by computing the distribution's mean force $\mathrm{PMF}(r)$.

$$U^{c.m.}(r) \approx \mathrm{PMF}(r) = -k_B T \log(g_{ex}^{c.m.}(r)) \tag{3.10}$$

$g_{ex}^{c.m.}(r)$ is the $\mathrm{RDF}_{c.m.}$, k_B the Boltzmann constant and T the temperature. The resulting potential is, however, only accurate for low simulation densities. Therefore, the PMF is only used as the starting potential and must be tuned until the RDF analysis and pressure of the Full Particle and Coarse Grain simulation match. Further details are in the papers by Praprotnik et al. [10].

For our Coarse Grain model, we allow multiple sites, thus not enforcing the usage of a single site. To map the full-resolution model to a Coarse Grain model consisting of many sites, we consult the MolMod database [15]. It contains models of various resolutions of many molecules based on empirical data. Therefore, we do not find new pair potentials but reuse the full-resolution potentials instead.

3.2. Coupling of Full Particle and Coarse Grain regions

When considering a scenario in which a Full Particle region is in direct contact with a Coarse Grain region, this change of resolution creates an artificial boundary. This may lead to a flux of molecules towards a particular direction or simulation failure because there is no smooth transition from one molecular representation to another. To remedy that, a Hybrid region (H) is introduced. In this area, molecules are simultaneously treated as a weighted combination of Full Particle and Coarse Grain. The weight ranges between 0 and 1 and is

defined as maximal when the molecule of interest is in a Full Particle region. It is minimal when the molecule is in a Coarse Grain region. [10]

Therefore, the force computation between two molecules α and β changes to:

$$F_{\alpha\beta} = w(r_\alpha)w(r_\beta) \cdot F_{\alpha\beta}^{FP} + (1 - w(r_\alpha)w(r_\beta)) \cdot F_{\alpha\beta}^{CG} \tag{3.11}$$

$$F_{\alpha\beta}^{X} = \sum_{i \in \text{site}(\alpha)} \sum_{j \in \text{site}(\beta)} F_{i\alpha j\beta} \tag{3.12}$$

with $F_{\alpha\beta}^{X}$ being the force between molecules α and β of resolution X, which is comprised of the sum of all pairwise site-site interactions. [10, 17]

Other implementations of the adaptive resolution scheme remove the rotational information for Coarse Grain and Hybrid molecules, as their CG representation consists of only a single site. This requires that once molecules cross the Hybrid Coarse Grain boundary, the rotational information must be reintroduced to maintain a constant level of latent heat. It is done by selecting a random FP molecule and copying its rotation. However, we do not remove the rotational DOF and thus omit this process. [12]

3.3. Weight Function

In literature, it is often not discussed how the actual molecule position correlates to the respective weight. Therefore, we compared different kind of implementations: `Euclid`, `Manhattan`, `Component` and `Nearest`. All versions first check if the point of interest is within the Full Particle or Coarse Grain region and return the edge values 1 or 0. The differences lie in the handling of points within the Hybrid region.
We define the simulation space as a cuboid in $\mathcal{R}^3$ in the first octant. A Full Particle region is a cuboid of arbitrary dimensions, and the associated Hybrid region is also a cuboid. The rest of the simulation domain is assumed to be of Coarse Grain resolution.

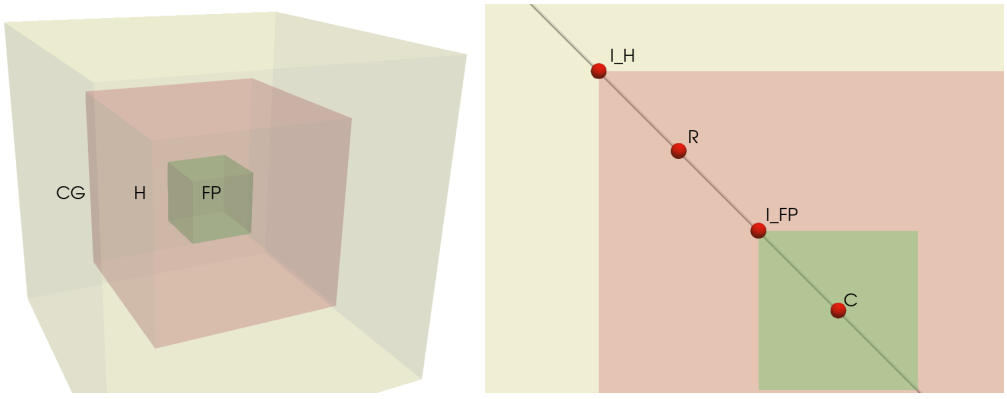


Figure 3.3.: Left: Region description for weight analysis. Yellow (Coarse Grain) Red (Hybrid) Green (Full Particle).
Right: 2D representation of the intersection of a line from the centre to the point of interest with the Hybrid and Full Particle borders for `Euclid` and `Manhattan`.

The handling of points R in Hybrid regions is done as follows. Our goal is to fit variations of cosine functions, such that they are monotonous and begin with one at the Full Particle region and end with zero at the Coarse Grain region. For Euclid, a line is created originating at the centre C of the Full Particle region and its direction vector is given by the vector from said centre point to the point of interest, RC. Next, the intersection points I_{FP}, I_H of this line with the Full Particle and Hybrid resolution cuboid are computed. With these points, we define the quarter period of the cosine function as the Euclidian distance between I_{FP}, I_H and the measured parameter as the Euclidian distance between I_{FP}, R:

$$b = \text{euc}(I_{FP}, I_H) \tag{3.13}$$
$$d = \text{euc}(I_{FP}, R) \tag{3.14}$$

Thus, resulting in the final formula:

$$w_{EUC} = \cos^2(\frac{\pi}{2b}d) \tag{3.15}$$
$$w_{EUC}(R) = \cos^2(\frac{\pi}{2b}\text{euc}(I_{FP}, R)) \tag{3.16}$$

The `Manhattan` version is identical in all aspects, apart from the used distance metric, which is the Manhattan norm:

$$b = \text{man}(I_{FP}, I_H) \tag{3.17}$$
$$w_{MAN}(R) = \cos^2(\frac{\pi}{2b}\text{man}(I_{FP}, R)) \tag{3.18}$$

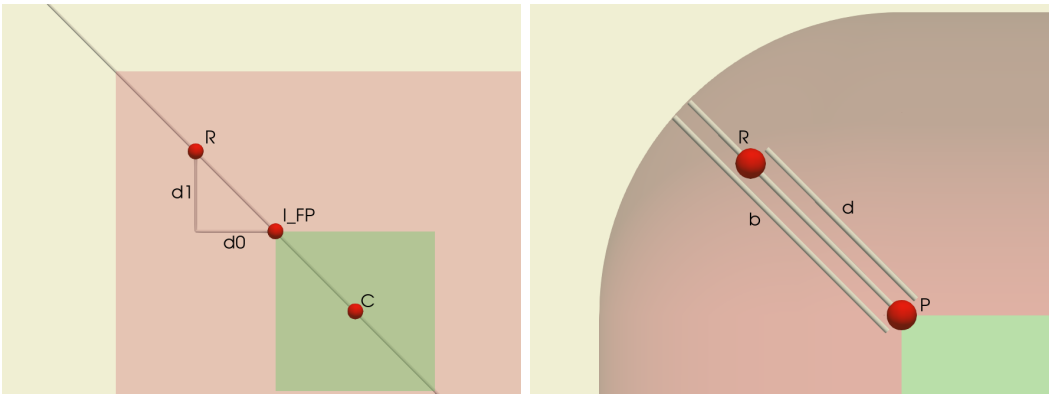


Figure 3.4.: Left: 2D distance description for `Component` version. In a 3-dimensional case, d_2 would be parallel to the vector, which is normal to the plane created by d_0 and d_1.
Right: Simplified visualization for `Nearest` using a 2D example. The Hybrid region has rounded corners. b is the full length from P to the edge of the Hybrid region and d is the distance from P to R.

The `Component`-based approach checks each component of the point of interest R individually. If R_i is within a Hybrid region, then a cosine function is applied to the distance R, which protrudes out of the Full Particle region defined by its bounding corners C_L and

C_H. The cosine quarter period length b is the width of the Hybrid region of the respective dimension i. The total weight is the product of the contribution of each dimension.

$$b_i = \text{hybrid}_i \tag{3.19}$$
$$d_i = \max(\max(C_{Li} - R_i, 0), \max(r_i - C_{Hi}, 0)) \tag{3.20}$$
$$w_i(R) = \begin{cases} \cos(\frac{\pi}{2b_i} d_i), & \text{if } R_i \text{ in hybrid} \\ 1, & \text{otherwise} \end{cases} \tag{3.21}$$
$$w_{\text{COM}}(R) = \prod_{i=0}^{2} w_i(R) \tag{3.22}$$

For the `Nearest` version, the Hybrid regions' bounding box is reinterpreted as a box with rounded corners and edges to reduce imbalances of the Hybrid width for these special cases. First, the point of interest R is projected onto a point P on the surface of the Full Particle regions' box, defined by C_L and C_H, such that the distance $RP = d$ is minimal. The cosine quarter period length b depends on whether R is on a side, edge or corner of the Hybrid regions' box. Therefore, we mix the involved lengths of the Hybrid shell by taking the square root of the sum of their square and use the resulting value as b.

$$b_i = \begin{cases} \text{hybrid}_i^2, & \text{if } R_i \text{ in hybrid} \\ 0, & \text{otherwise} \end{cases} \tag{3.23}$$
$$b = \sqrt{\sum_{i=0}^{2} b_i} \tag{3.24}$$
$$d_i = \max(\max(C_{Li} - R_i, 0), \max(r_i - C_{Hi}, 0)) \tag{3.25}$$
$$d = \sqrt{\sum_{i=0}^{2} d_i} \tag{3.26}$$
$$w_{\text{NEA}}(R) = \begin{cases} \cos^2\left(\frac{\pi}{2b} d\right), & \text{if } d < b \\ 0, & \text{otherwise} \end{cases} \tag{3.27}$$

In order to find the best choice for the weight function, we created a scenario in which Cyclohexane (C_6H_{12}) molecules in a domain of $200 \times 200 \times 200$ are first equilibrated for 10,000 time-steps using only Full Particle resolution. Then the simulation was restarted while using AdResS with the different weight functions. We also added one weight implementation that emulates a missing weight function. It treats all Hybrid molecules as pure Coarse Grain ones, thus introducing a harsh boundary between FP and CG. Each simulation ran for 100,000 time-steps, and the end states are illustrated in figure 3.5.

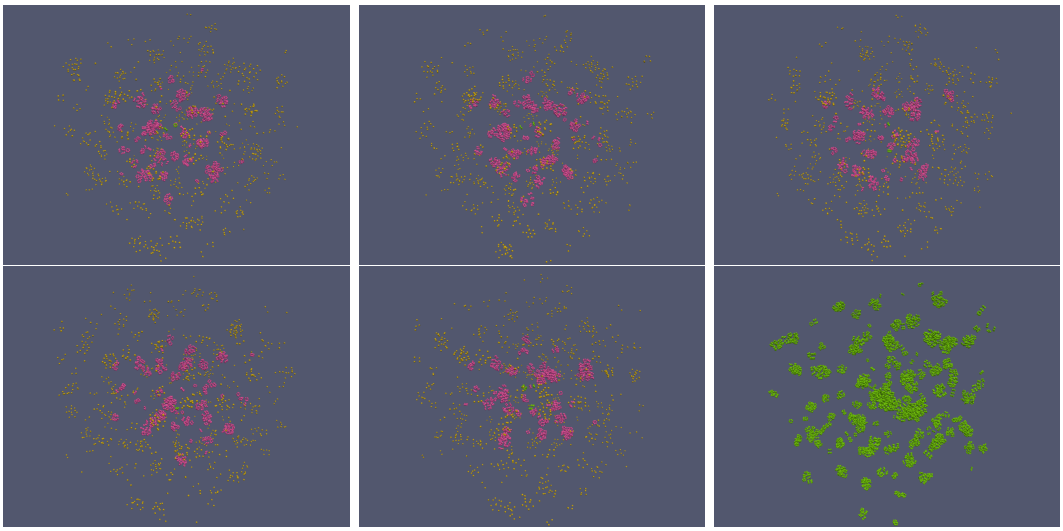


Figure 3.5.: Comparison of the end state of a simulation with C_6H_{12} using different weight functions after 100,000 time steps. Description from top left to bottom right: used weight function 1) Euclid 2) Manhattan 3) Component 4) Nearest 5) No weight 6) No AdResS. Colour Description: green = Full Particle, pink = Hybrid, yellow = Coarse Grain

It is worth mentioning that we encountered many thermostat warnings during the run with no weight. This is most likely due to molecules changing the resolution in close proximity without a smooth transition, therefore temporarily experiencing high forces. Molecules did not go out of bounds. However, it is possible for this to happen in other scenarios. To prevent that, it is necessary to use a weight function. All end states were analyzed regarding density distribution with a tool discussed in section 8.2. With the resulting densities, we computed the relative error against the reference run with no AdResS. This was done for every sample point in the domain. We used the sum of the errors across the entire region and the Full Particle region as a quality metric. The resulting errors are displayed in table 3.1. As the nearest weight function was the best in both cases, we used this in our further studies.

Region	EUC	MAN	COM	NEA	OFF
All	48313	62347	58146	45050	50729
FP	-46.7940	-32.0643	-4.3000	-0.5000	-60.3488

Table 3.1.: Overview of the sum of relative error in either full domain or Full Particle region for each implementation. The relative error is measured against the reference implementation.

4. Related Work

4.1. Grand Canonical AdResS (GC-AdResS)

When using AdResS, differences in chemical potential often occur in the boundary region of two different resolutions for a single molecule type, leading to an unexpected density change. Such behaviour is observable in our tests in section 8.2. By introducing a position-dependent thermodynamic force, which acts as a correction force to achieve a correct equilibrium state, the simulation can be considered Grand Canonical in a thermodynamic sense. It can be argued that the CG region acts as a buffer or reservoir with which the FP region exchanges energy and particles. [9, 18]

4.2. Open Boundary MD

Open boundary simulations, in which domain bounds are not periodic, but the domain exchanges molecules with a surrounding system, can be achieved using AdResS. The reasoning behind this is that AdResS can gradually change resolutions and therefore introduce arbitrary molecules, which do not exert high forces in the CG region. Furthermore, it can be viewed as an alteration to GC-AdResS in which the GC region is given the role of an interface to the outside buffer. This approach, OBMD, can be extended to fluid dynamics (CFD) in the outer region, in which, e.g., Navier-Stokes equations define the movement of fluid. [9]

4.3. Hamiltonian AdResS

In common AdResS, Newton's third law is enforced throughout the simulation. Assuming a force-based approach, weights are applied to the interacting forces. This prevents the formulation of an interpolation scheme for potential energy using a function depending on the position. Therefore, the accuracy of the computed potential energy is reduced. One mitigation method is using a global Hamiltonian function while relaxing the constraint. This is known as H-AdResS. [9, 8, 4]

Part II.

Implementation in ls1-Mardyn

As the main topic of this thesis is the exploration of AdResS in ls1-Mardyn, this part focuses on the implementation. First, we briefly introduce the architecture of ls1-Mardyn and the relevant aspects regarding AdResS. Since we did not design ls1-Mardyn from the ground up, design decisions about how to add this new feature need to be clarified. Then we show our actual implementation and evaluate it considering accuracy and performance. The evaluation uses different simulations, which are described in full detail.

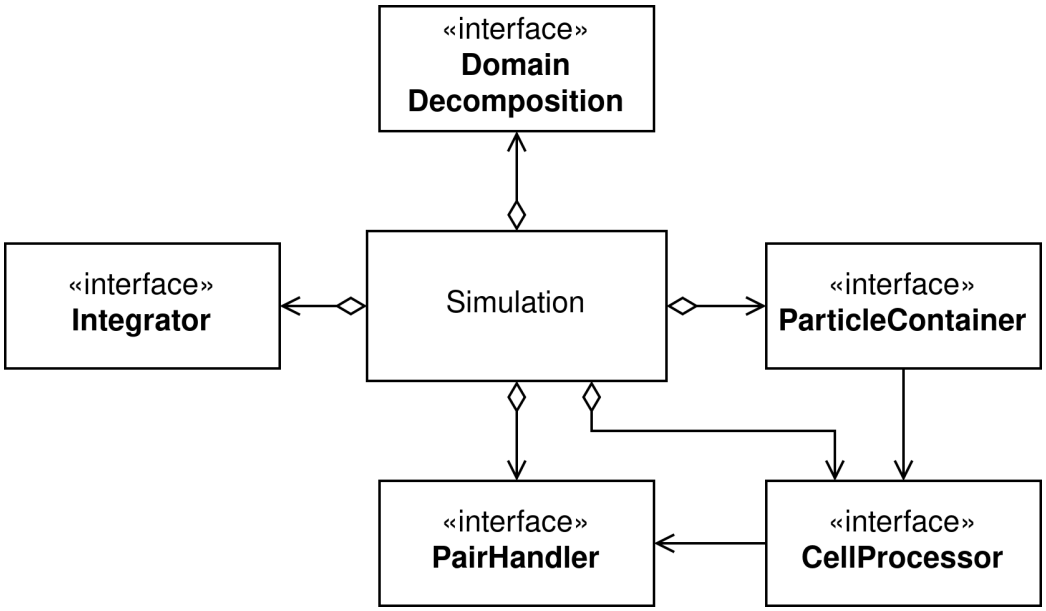


Figure 4.1.: Simplified overview of ls1-Mardyn architecture as UML diagram.

5. ls1-Mardyn Architecture

Like the majority of MD-Simulation software, ls1-Mardyn has the following components: `Simulation`, `Particle Container` and `Integrator`. `Cell Processor` and `Pair Handler` are used for traversing all molecules and molecule pairs generically. In addition, an event system for plugins is implemented, in which every plugin is notified at specific points during each time step. A structural overview is given in the UML diagram in figure 4.1.

5.1. Simulation

The `Simulation` class is the main component that ties everything together. It manages memory for most of the other components and primarily handles the simulation loop. In a simplified view, the main loop first initialises the new time-step and all related events. Then

the `Particle Container` is updated. This includes `Domain Decomposition` and optional load rebalancing. Then all forces are computed. In the end, an integration step is performed, and thermostats are applied. A more in-depth description of the simulation loop is shown in figure 5.1.

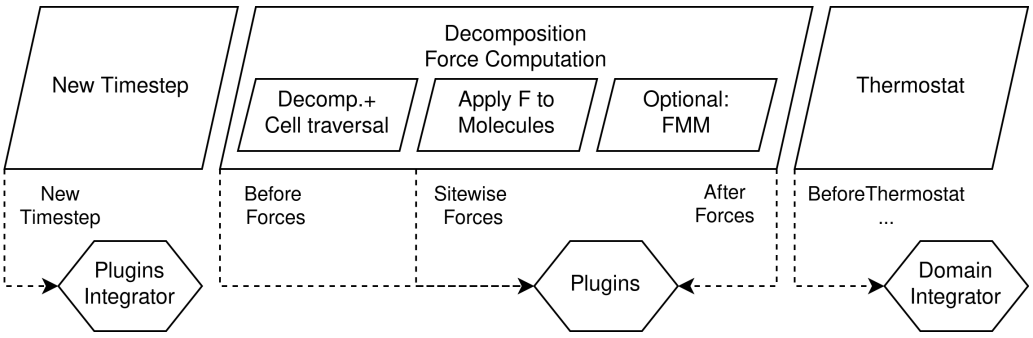


Figure 5.1.: More detailed diagram of all steps within a single pass of the main simulation loop. The parallelograms describe function calls from `Simulation`. The arrows indicate with their position the time when an event is created in `Simulation`. The hexagons are the receivers of the events.

5.2. Domain Decomposition

Because ls1-Mardyn can run on multiple nodes, the workload must be split across those. This is done by `Domain Decomposition` by dividing the simulation space into smaller chunks and letting each node simulate one of those subdomains. `DomainDecompBase` is the general interface enabling this feature. Different options are selectable at runtime. `DomainDecomposition` splits the simulation space equally and does not provide any load-balancing functionality. `GeneralDomainDecomposition` splits the space according to the timing information of each node using the ALL library in order to mitigate imbalances. `KDDecomposition` rebalances by constructing a KD-Tree based on the molecule counts of each node.

5.3. Integrator

To advance time in ls1-Mardyn, integrators are used. These update the velocity and position of all molecules. The implemented `Integrator` is based on the leapfrog algorithm. Therefore, the integrator needs to be called twice per simulation step, once in the beginning and once after the force computation.

5.4. Particle Container

Molecules are stored in a `Particle Container` throughout the simulation. This interface provides functionality for molecule insertion and extraction as well as general methods to traverse the data structure. All implementations are assumed to split the domain into cells, such as in the Linked Cells and Verlet Lists algorithm. Therefore, the interface

provides methods that traverse all cells based on the selected strategy and use the provided `Cell Processor` to handle the individual cells and cell pairs. The implemented `Particle Containers` are `LinkedCells`, which implements the algorithm of the same name, and the `AutoPas` container, which can either use Linked Cells or Verlet Lists internally and also change the traversal strategy during runtime for better performance.

5.5. Cell Processor

The `Cell Processor` interface allows for an initialisation and end step for each traversal. During traversal methods that process cell pairs, single cells or similar ones are called. ls1-Mardyn uses these `Cell Processors` in multiple ways. The main use case is for force calculation. The `LegacyCellProcessor`, which does not use vectorization, the `VectorizedCellProcessor` and `VCP1CLJRMM` are such examples. The latter also uses vectorization but is utilised in reduced memory consumption mode. However, `Cell Processors` can also be used to gather auxiliary information like in the `FlopCounter`, `ODFCellProcessor` or `RDFCellProcessor`.

5.6. Pair Handler

As the `LegacyCellProcessor` does not use vectorization, interactions of molecule pairs do not directly use the backing data fields but rather the abstraction layer provided by the `Molecule` interface. This is done by the `PairHandler` interface.

6. Particle Container Design

With ls1-Mardyn being a software many years in development with a large codebase, adding AdResS required several design decisions. Critical factors were encapsulation, potential performance, ease of implementation and invasiveness into existing code.

6.1. Approach: 1 Particle Container

The initial idea in this approach was to use a single particle container, use the existing plugin support and not change any other code. We succeeded in doing so by writing a plugin which follows the given procedure:

1. It is called the first time before force computation. During this step, all molecules are traversed, and it is checked if the resolution of the molecule must be changed.
2. The second call occurs after the ls1-Mardyn native force components compute all forces. The plugin then had to undo all computed forces, which involved Hybrid molecules, because the native implementations were not aware of the AdResS molecule model and weight functions.
3. As the next step, the plugin recomputed those interactions with respect to AdResS.

Since an AdResS-aware `ParticlePairHandler` was implemented for the next approach, we decided to reuse it in this approach. This could only be done by setting the `CellProcessor` in the `Simulation` object to a `LegacyCellProcessor`, which uses our `ParticlePairHandler`. Due to the design choice of keeping changes to existing code at a minimum, this reassignment must be done in the AdResS plugin. Therefore, a single setter for the `CellProcessor` was added to the `Simulation` class. By doing so, all forces are already computed correctly during the native `ParticleContainer` traversal, resulting in no further need for any force computation in the plugin.

6.2. Approach: 3 Particle Container

The alternative was to use three particle containers, one for each AdResS resolution. Here, molecules belonging to Full Particle, Hybrid and Coarse Grain resolution are in their respective containers. This idea was pursued because, at the time of design, the initial approach did not use the `ParticlePairHandler` and required the recomputation of forces. We created the three-container approach to eliminate that and increase the ease of tracking individual resolutions.
Major refactoring was necessary as ls1-Mardyn was not designed to use multiple `ParticleContainers`:

1. For the force calculation, three `CellProcessors` and `Integrators` were created, one for each container.
2. For the `CellProcessors` responsible for the Full Particle and Coarse Grain resolution, we could reuse the `VectorizedCellProcessor`, as no information about AdResS is needed for those molecules.
3. The Hybrid container required the development of a `ParticlePairHandler`, which was used by a `LegacyCellProcessor`.
4. A plugin was still developed because AdResS functionality could be enabled or disabled at runtime. The plugin handled the remaining interactions of molecules between the different `ParticleContainers`. It also moved molecules between containers when the molecule resolution changed.

6.3. Evaluation

The first approach was much easier to implement, as it did not require many changes to existing code. Therefore, using one particle container is better regarding ease of implementation and invasiveness into existing code. Considering encapsulation, the first approach only required the addition of a setter of the `CellProcessor`. However, the second approach required many other changes, which made it worse in this aspect.

In our preliminary investigations, the three-container version was not faster than the other approach in any scenario. The main reason for that is most likely the suboptimal parallelization method used for the force computation in the plugin and the inherent overhead of traversing the simulation domain multiple times. We picked a slicing approach to compute forces in parallel for ease of implementation due to the lack of regional cell-based iteration across `ParticleContainers`. Furthermore, moving data from one container to another also decreased performance.

Additionally, scaling across multiple nodes was worse, as more overhead and communication were created due to the higher amount of containers. It should also be noted that we achieved high but not complete physical correctness with three containers. We assume that this is due to a bug in the code, which we could not find. Because ls1-Mardyn was not designed to use three `ParticleContainers`, finding errors was very time-consuming. Hence we decided to discontinue the three-container version and only work with the first approach.

7. AdResS Implementation

In the previous chapter, we discussed our approach and how it is designed. In the following sections, implementational details are explained.

7.1. Data Representation

The adaptive resolution scheme requires information about the regions where a certain resolution should be used and some method to represent the different molecule resolutions. Regional data is stored in a vector of FPRegion objects. These represent a box for the full-resolution region by storing the lower and upper bounds. Additionally, they contain information on the width of the Hybrid region, which is a shell around the full-resolution box. The size of the Hybrid shell may vary in each dimension. The FPRegion struct in listing 7.1 also provides functions to compute intersections.

```
1 struct FPRegion {
2   // FP box
3   std::array<double, 3> _low;
4   std::array<double, 3> _high;
5   // H box
6   std::array<double, 3> _lowHybrid;
7   std::array<double, 3> _highHybrid;
8   // ... methods
9 };
```

Listing 7.1: FPRegion struct.

For force and weight computation, we require some auxiliary functionality. This is described in the following enumeration:

1. `FPRegion::isInnerPoint` checks if a provided point is within a specified box, defined by its lower and upper bounds.
2. `FPRegion::computeIntersection` computes the intersection point of a line, whose directional vector is defined by a provided point in 3D space and the centre of the FPRegion box, with either the box that defines the full-resolution region or the outer shell for the Hybrid region depending on the input parameter.
3. `FPRegion::isRegionInBox` checks if the box representing the Full Particle area is entirely inside a provided box, defined by its lower and upper bounds.
4. As ls1-Mardyn uses periodic bounds, we need to handle that. Users need not consider edge cases when specifying an `FPRegion` that crosses domain bounds in an input file. That pushes the responsibility to the implementation. Therefore, `FPRegion` boxes wrap around the simulation space when they cross domain bounds. To check whether a given point is within either one of the boxes defined by an `FPRegion` object that is potentially

wrapped around the simulation domain, `FPRegion::isInnerPointDomain` wraps the region in all possible forms across domain bounds, computes the boundary coordinates of all boxes created after wrapping and checks with `FPRegion::isInnerPoint` if the provided point is within any of these boxes.

In ls1-Mardyn, all molecules are stored in the `Particle Container`. For higher memory efficiency, the molecular structure and site information is not stored in every molecule but instead in a global `Component` object, which all `Molecule` instances point to. Therefore, a global vector contains one `Component` object for each molecule type. In our implementation of AdResS, we reused this and did not implement a new `Molecule` class for the different resolutions. Instead, we define a new `Component` for the three resolutions for each molecule type. The resolution change is done by updating the `Molecule` object's `Component` pointer. The `Component` vector is indirectly owned by the `Simulation` object.

7.2. Plugin

The plugin class bundles all AdResS-related features together. It contains:

1. Pointer to the `ParticlePairsHandler`
2. Vector of all `FPRegions`
3. Pointer to the `Particle Container`
4. Pointer to the vector of all `Component` instances
5. Mapping from `Component` IDs to their respective resolution
6. Pointer to the globally valid `Domain` object
7. Buffer for macroscopic value computation
8. Pointer to the selected weight function

All weight functions are implemented in this class. The weight implementation is selected while reading the simulation's configuration file, and `FPRegions` are constructed and stored in the buffer. During this step, the `PairHandler`, `CellProcessor`, and optionally `DomainDecomposition` are set accordingly. In later initialisation, the mapping from `Component` ID to resolution is set up, and the remaining fields are initialised.

The event call `beforeForces` traverses all molecules and checks if their resolution has changed. This check must be done at this event because, after earlier events, the `Integrator` moves molecules, which can invalidate the current resolution. At later points, forces are computed, which require the correct molecular representation. If a molecule moves, the level of detail can change, which may require an update of the `Component` pointer. `AdResS::checkMoleculeLOD` does this by comparing the molecules mapped `Component` ID to a specified target resolution. A simple offset is computed to acquire the correct component pointer as `Components` of the same molecule type are inserted into the global vector in the order FP, H, CG.

7.3. Force Computation

All forces are computed during the traversal of the `Particle Container`. As understanding and reimplementing the `VectorizedCellProcessor` went beyond our scope, we reused the `LegacyCellProcessor`. This, in turn, uses the `AdResSForceAdapter`, which implements the `ParticlePairsHandler` interface, to compute forces between individual molecules. The interface defines the functions: `init`, `finish` and `processPair`. The first two manage buffers for multi-threading. The last is an adapter between the `Cell Processor` and the internal implementation. The force computation is delegated to the respective backend, which depends on the type of molecules. Interacting molecule pairs without Hybrid molecules can be handled by the native implementation. Pairs with Hybrid molecules must be handled according to the AdResS force computation. If only one molecule is Hybrid, then simplifications apply. We explain the functionality of these backends based on `AdResSForceAdapter::potForceFullHybrid` as the other functions are either the native implementation or minor variations of this function.
According to equation 3.12, all sites of full resolution interact with each other, resulting in one part of the total force. This part must be multiplied by the product of the weights. The analogue is true for the Coarse Grain sites, except for the inverted weight.
These exact steps are followed in the function. First, all LJ sites are processed. It is checked if both sites are either full resolution or Coarse Grain (exclusive). Then the potential and force between those two sites are computed. The appropriate weight is multiplied by the partial force and potential depending on the resolution. This is done at this point, as the cumulative force for one molecule is the sum of the forces of all sites. Therefore, the weight factor can be distributed into each term. By doing so, we can reuse most of the code of the native `ParticlePairsHandler`. Other site combinations are skipped. The same is done for all polar site combinations. An example is given in listing 7.2.

```
1  wi = AdResS.weight(Mol1);
2  wj = AdResS.weight(Mol2);
3  for si in 1...Mol1.sites() do
4    for sj in 1...Mol2.sites() do
5      if Mol1.site(si).isCoarseGrained()
6         xor Mol2.site(sj).isCoarseGrained() then
7        skip
8      endif
9      Force, Potential = computePot(Mol1, Mol2, si, sj);
10     if Mol1.site(si).isCoarseGrained() then
11       Force *= 1 - wi * wj;
12       Potential *= 1 - wi * wj;
13     else
14       Force *= wi * wj;
15       Potential *= wi * wj;
16     endif
17
18     Mol1.addForce(Force, si);
19     Mol2.subForce(Force, sj);
20   end
21 end
```

Listing 7.2: Pseudo code for generic potential force calculation.

8. Accuracy and Performance

8.1. Benchmark Scenario

To evaluate our implementation in terms of accuracy and performance, we constructed different benchmark scenarios focusing on different aspects. For all simulations, we used an interface scenario of oxygen (O_2) with either a larger cyclohexane (C_6H_{12}) or a smaller methane (CH_4) combustible molecule. This was based on the studies by Nitzke et al. [7]. This means all simulation domains have a similar structure. The left half of the domain is filled with O_2 and the right half with fuel. Properties such as domain size, Full Particle region position and Hybrid dimensions vary between the different scenarios. In all configurations for the molecular representation of C_6H_{12}, we used a 6-site model for FP and a single site for CG. For CH_4, we used the same single site model for CG and FP because the MolMod database [15] did not contain alternatives. The O_2 molecule contained five sites in FP and three in CG. Based on these, we initialised all simulations according to the following steps:

1. Create a layer of O_2 and the fuel in separate simulations.
2. Let those equilibrate in 10,000 time-steps only using Full Particle resolution.
3. The main configuration uses the two resulting phase-space files and merges them into one larger simulation.

The runs to determine accuracy were set to 100,000 time-steps, whereas all others used 10,000 steps.

Scenario: BASE

We have two main kinds of scenarios. In the first (BASE), we created a cubic domain and used molecule counts ranging from 4.0e+3 to 1.0e+8. We also matched the system's pressure to 5 MPa for C6O12 and 0.4 MPa for CH_4.[7] As the number of molecules altered, the domain size was adjusted such that the pressure remained approximately constant while retaining a cubic shape. AdResS was set up such that in the middle of the domain, where the different molecules interface, is a slice of width 40 as the Full Particle region. On the left and right is a slice of width 40 for the Hybrid region. The remaining area of the domain is Coarse Grain. Figure 8.1 gives an example demonstrating the proportions in x-Axis. These simulations were run for accuracy and performance with fixed Full Particle and Hybrid region width but with varying node counts.

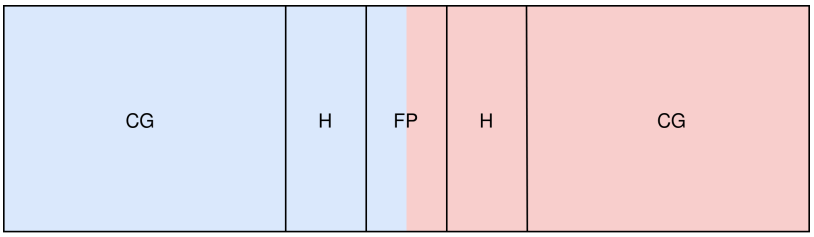


Figure 8.1.: Schematic 2D representation of the BASE scenario. Blue represents the oxygen and red the fuel. The dimensions are exaggerated for representation purposes. This horizontal subsection of the cube should illustrate the general structure. CG: Coarse Grain H: Hybrid FP: Full Particle

Scenario: HYBRID

To determine the effects of the Hybrid region on performance and accuracy, we created another scenario (HYBRID) by changing the BASE run that had 1.0e+6 molecules. The domain size and molecule densities are unchanged. The only variable is the width of the Hybrid region. As before, we have a slice of width 40 in the middle as the Full Particle region. The Hybrid slice's width was set to 10% to 90% of the length of the distance from the beginning of the simulation domain to the beginning of the FP region along the x-axis. Therefore, the only changes occur in the AdResS configuration. This is shown in figure 8.2.

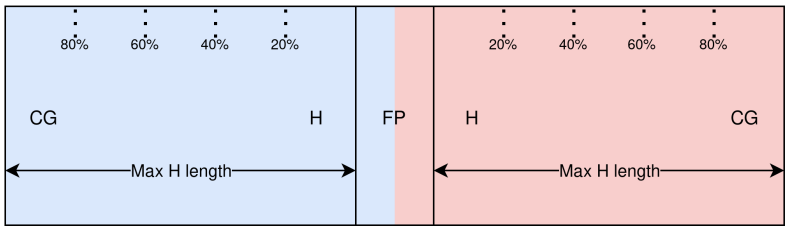


Figure 8.2.: Schematic 2D representation of the HYBRID scenario. Blue represents the oxygen and red the fuel. The dimensions are for representation purposes. This horizontal subsection of the cube should illustrate the general structure. The maximum Hybrid length begins at the start of the domain and ends at the start of the Full Particle region. The Hybrid size used for the individual simulations ranges from 10% to 90% of that length. The remaining area on the outer sides has Coarse Grain resolution. CG: Coarse Grain H: Hybrid FP: Full Particle

Scenario: XSCALE

The pressure was neglected in the second kind (XSCALE) but was still approximately accurate as the initialisation density did not change. Here the focus was on the effects of increasing the simulation domain while changing the proportion of Full Particle and Hybrid regions to Coarse Grain regions. This was considered since the interface scenario does not reflect a common use case regarding the proportion between CG and FP molecules. Examples are simulations of solvation processes of proteins in water, in which the Full Particle and Hybrid region are only a small fraction of the entire domain, while the majority

is Coarse Grain. [14, 9] The domain structure is similar to the prior simulations. In the middle is the slice of width 40 as the Full Particle region, which is surrounded by slices of equal size left and right as the Hybrid region. The changing property here is the size of the Coarse Grain region, which ranges from 100 to 1,000 units on either side. This is shown in figure 8.3.

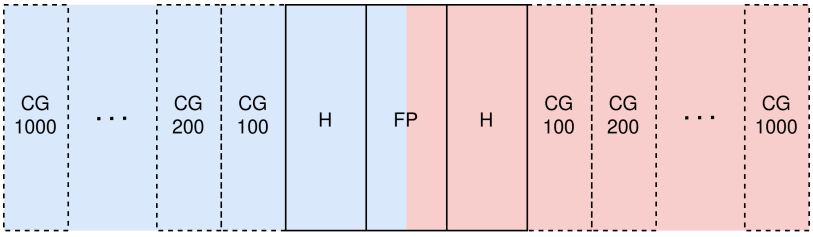


Figure 8.3.: Schematic 2D representation of the XSCALE scenario. Blue should represent the oxygen and red the fuel. The dimensions are for representation purposes. Here the Coarse Grain region is scaled from 100 units up to 1,000. CG: Coarse Grain H: Hybrid FP: Full Particle

8.2. Accuracy

We analysed the accuracy by generating output in the vtk format, on which we applied a simplified RDF function. Our analysis program first splits the domain into bins. This can be done along one or any selection of the dimensions. For the following accuracy analysis, we split along the x-axis. Afterwards, the partial density of each molecule type is computed for each bin by counting the respective molecule type count and then dividing by the bins' volume. We did this analysis for each run, with and without AdResS. Afterwards, we computed the difference between the AdResS bin densities to the reference.

First, we discuss the results of the BASE scenario. The densities of the CH_4 runs are displayed in figures 8.4, 8.5. In the runs, AdResS and Reference, with 4.0e+3 molecules, the density distribution of both molecule types display a random behaviour with no discernible pattern. With 1.60e+4 molecules, an interleaved cosine structure starts to appear. This persists with higher numbers of molecules. Overall, the AdResS results are quite similar to the reference ones. Especially the interface region, which was in full-resolution, seems to follow the same trajectory. One issue is that for 2.56e+5 and 1.0e+6 molecules, the O_2 density drops and rises rapidly precisely at the boundary point in the AdResS runs. This behaviour can also be seen with a smaller amplitude in the reference's density distribution for 2.56e+5 molecules, which leads to the assumption, that such fast changes in curvature in the reference density are exaggerated through our AdResS implementation in the Full Particle region.

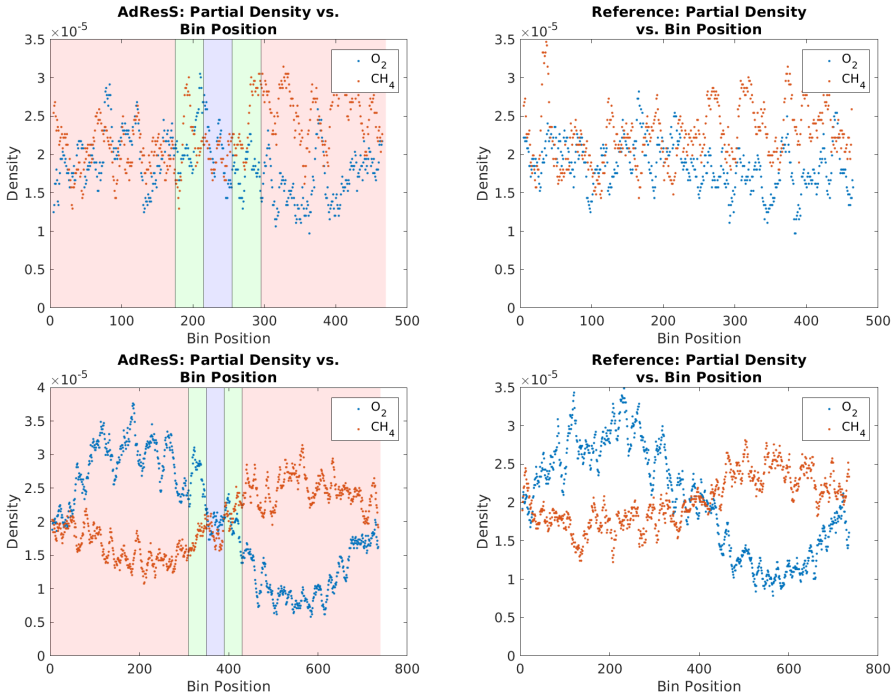


Figure 8.4.: Density distribution of CH_4 runs with 4.0e+3 and 1.60e+4 molecules (ordered top to bottom). Left column: AdResS, Right column: reference. Background: Red = Coarse Grain, Green = Hybrid, Blue = Full Particle

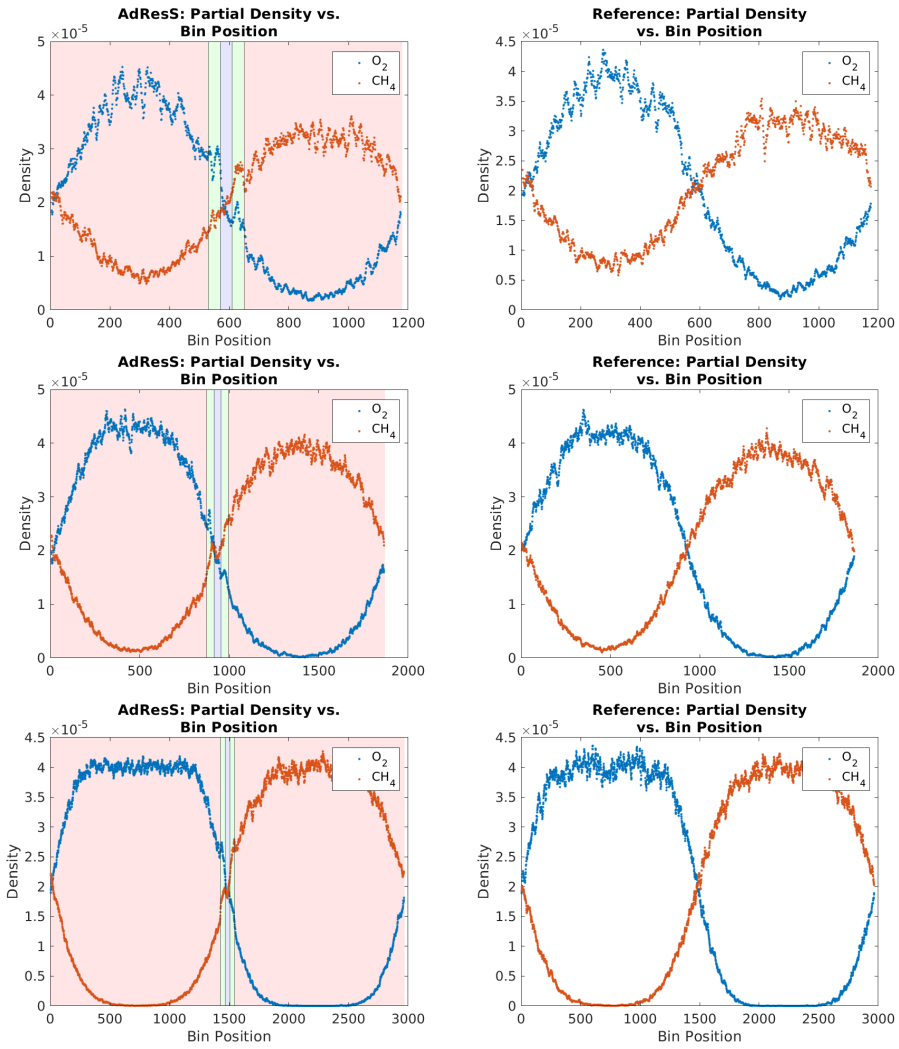


Figure 8.5.: (Density distribution of CH_4 runs with 6.40e+4 through 1.0e+6 molecules (ordered top to bottom). Left column: AdResS, Right column: reference. Background: Red = Coarse Grain, Green = Hybrid, Blue = Full Particle

The difference between the AdResS and reference data, displayed in figure 8.6, also suggests that the accuracy increases with rising molecules count. For 4.0e+3 and 1.6e+4 molecules, the difference is in the same order of magnitude as the actual densities, indicating a high deviation. However, starting with 6.40e+4 molecules, the difference becomes less and for 1.0e+6 molecules, it is by one order of magnitude less in the Full Particle region. We presume the relatively high accuracy is linked to the used molecule model, as the CH_4 molecule was not affected by AdResS. At this point, it is worth mentioning that the empty space in the difference graph of the 1.0e+6 run is due to the nature of our analysis. When not both of the densities for a bin of the AdResS and reference run were non-zero, we removed that data point.

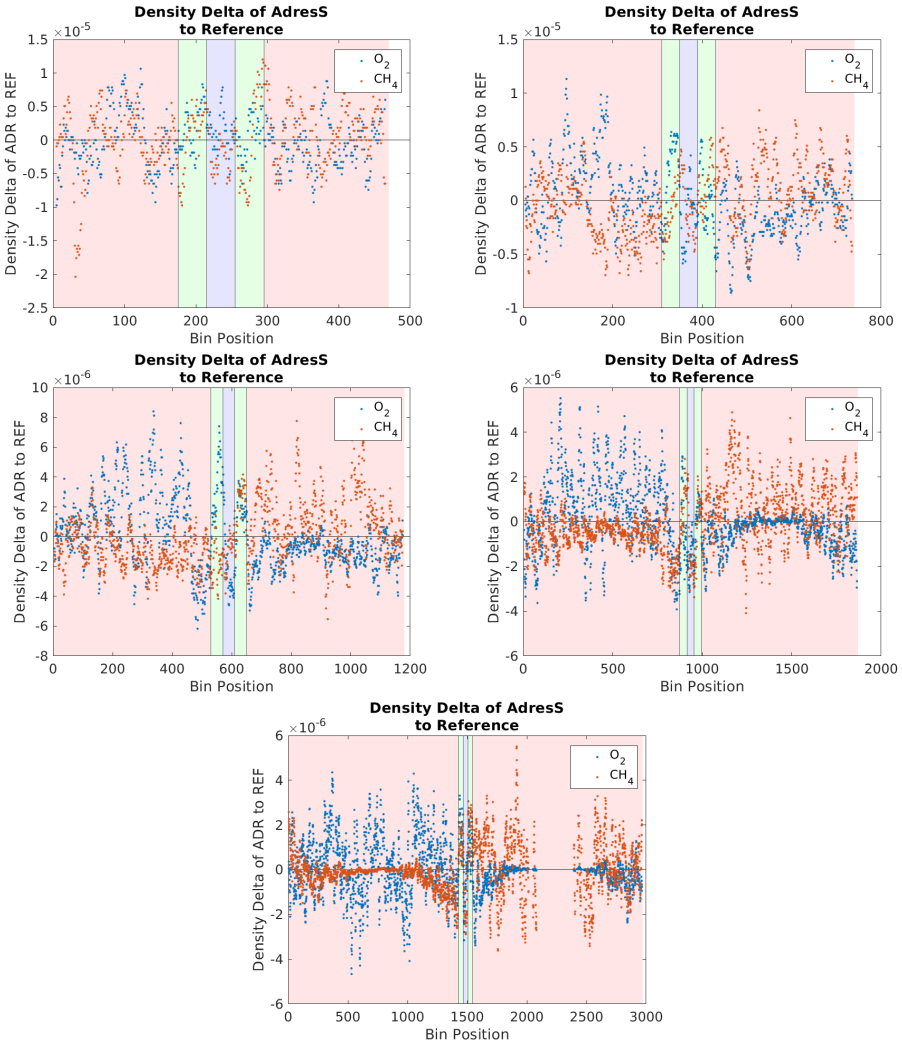


Figure 8.6.: CH_4: Difference of densities between AdResS and reference, computed at positions where both inputs had densities not equal to zero. This was done to see the error and not input data. Background: Red = Coarse Grain, Green = Hybrid, Blue = Full Particle

The results of the C_6H_{12} runs are displayed in figures 8.7 and 8.8. When comparing the graphs for each molecule count, the accuracy increases with higher particle counts but starts with significantly higher differences. The AdResS results show strong deviations from the reference for lower molecule counts. The run with 4.0e+3 molecules has a "spiky" profile in the reference. This is also reflected in the AdResS runs. However, the amplitude is approximately twice as large. Similar to the CH_4 run, the interleaved cosine structure starts appearing in the reference run with 1.6e+4 molecules. However, this is not apparent while using AdResS. To a certain extent, it only displays some correlation to the reference.
Additionally, the high peaks around the borders of the Full Particle region contribute to an error high enough such that the density profile in the interface region is unusable. Starting with 6.4e+4 molecules, the amplitude of the peaks is reduced far enough such that the

interface's density profile is comparable. However, the same change in curvature in the middle as the CH_4 runs is observable. This trend continues with more molecules. As the density profiles stabilize and show the interleaved sine-wave pattern for 6.4e+4, 2.56e+5, and 1.0e+6 molecules, it becomes apparent that the interactions between CG C_6H_{12} and CG O_2 result in a drop in the density of C_6H_{12} in the CG region. In the FP region, the interface density profile has higher similarity and follows the same trajectory. Only the absolute density is slightly lower. This is likely due to the high differences in the C_6H_{12} representation. We attribute the differences between AdResS and reference to dissimilar molecular representations, even though those were taken from the MolMod database [15].

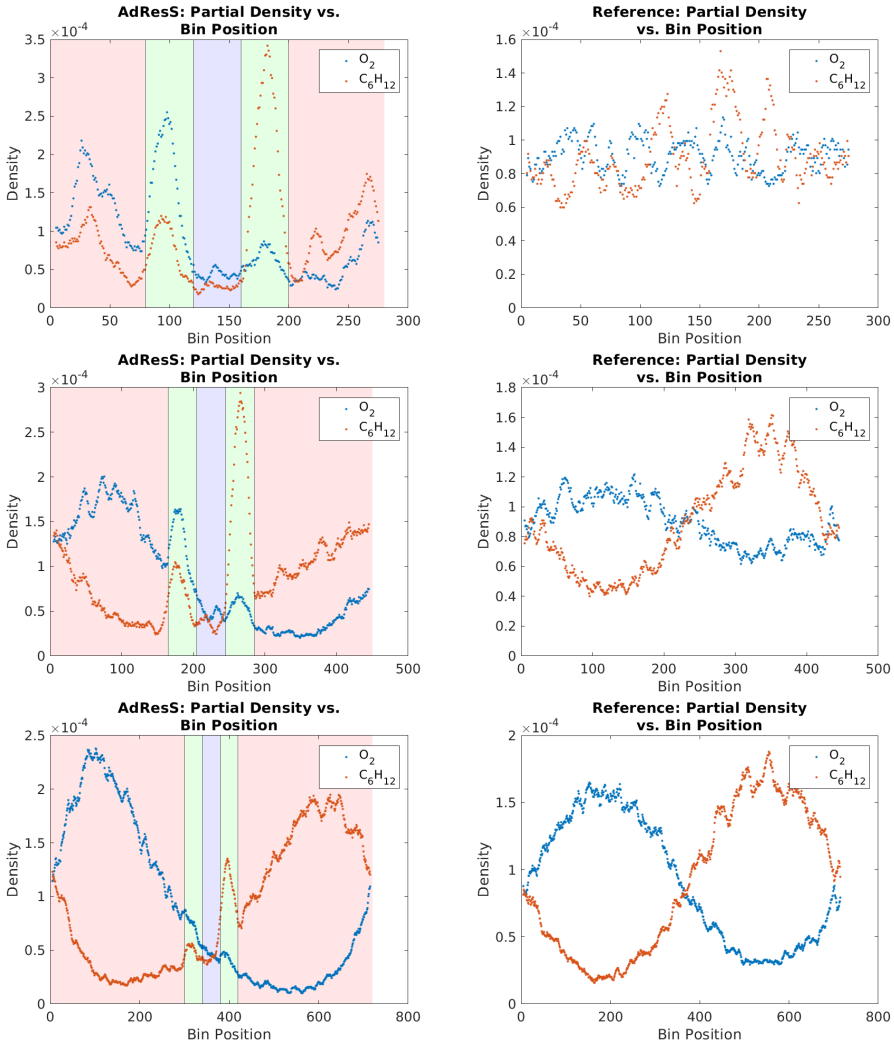


Figure 8.7.: Density distribution of C_6H_{12} runs with 4.0e+3, 1.6e+4, and 6.40e+4 molecules (ordered top to bottom). Left column: AdResS, Right column: reference. Background: Red = Coarse Grain, Green = Hybrid, Blue = Full Particle

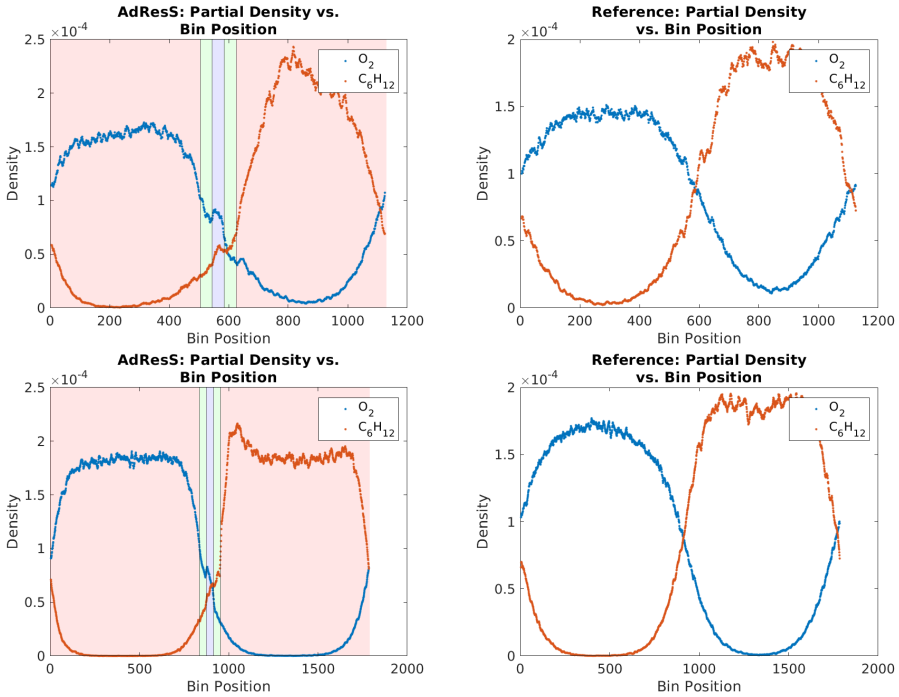


Figure 8.8.: Density distribution of C_6H_{12} runs with 2.56e+5 and 1.0e+6 molecules. Left column: AdResS, Right column: reference. Background: Red = Coarse Grain, Green = Hybrid, Blue = Full Particle

The findings mentioned for the difference between AdResS and reference density for CH_4 also apply to C_6H_{12}, as displayed in figures 8.9 and 8.10. Additionally, the "spikes" are also visible in the C_6H_{12} differences. Also problematic is that high differences exist precisely in the Full Particle region. Therefore, without further adjustments, we conclude that it is not viable to directly query the MolMod database [15] for different molecular representations concerning large molecules. Either further tweaking is needed or the regular approach of sampling a new potential function must be undertaken. This may work for small molecules such as O_2 or functional groups or similar size.

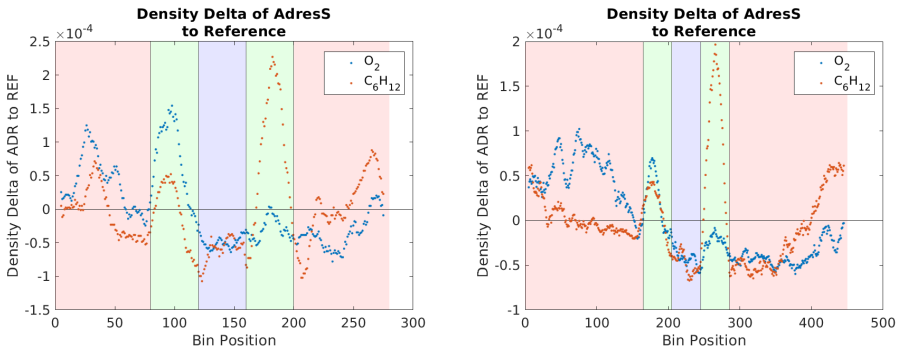


Figure 8.9.: C_6H_{12}: Difference of densities between AdResS and reference, computed at positions where both inputs had densities not equal to zero. Background: Red = Coarse Grain, Green = Hybrid, Blue = Full Particle

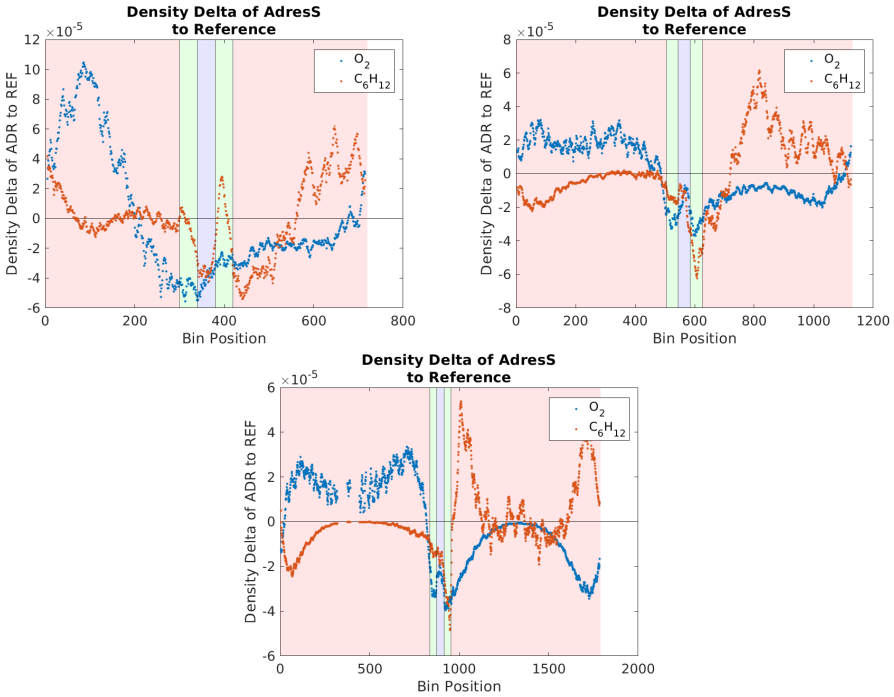


Figure 8.10.: C_6H_{12}: Difference of densities between AdResS and reference, computed at positions where both inputs had densities not equal to zero. This was done to see the error and not input data. Background: Red = Coarse Grain, Green = Hybrid, Blue = Full Particle

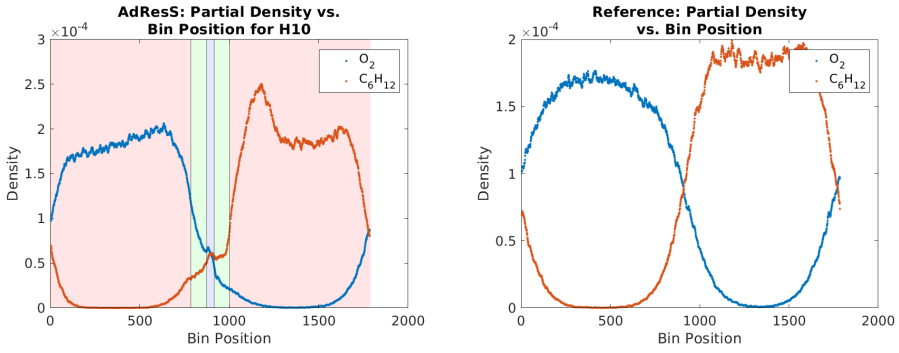


Figure 8.11.: Density distribution of HYBRID C_6H_{12} runs with 10% Hybrid width (ordered top to bottom). Left column: AdResS, Right column: reference. Background: Red = Coarse Grain, Green = Hybrid, Blue = Full Particle

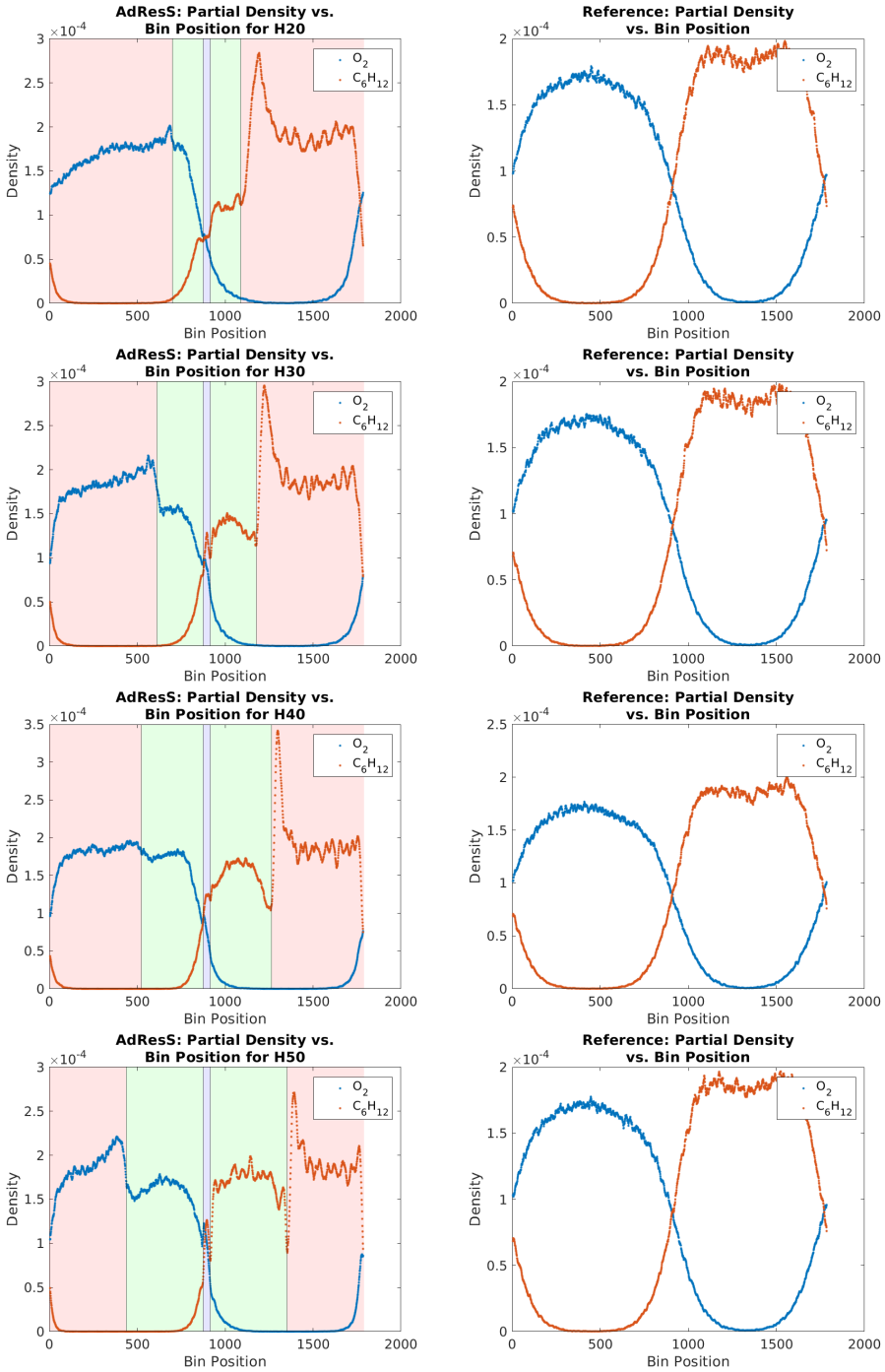


Figure 8.12.: Density distribution of HYBRID C_6H_{12} runs with 20% to 50% Hybrid width (ordered top to bottom). Left column: AdResS, Right column: reference. Background: Red = Coarse Grain, Green = Hybrid, Blue = Full Particle

Further insight into accuracy is given by checking the HYBRID runs, which are shown in figures 8.11-8.14. Here we see the effects of the Hybrid width on accuracy. As the C_6H_{12} BASE runs displayed higher differences, we only focus on the C_6H_{12} HYBRID runs. The graph with 10% H-width shares the highest similarities with the BASE scenario because the width of the Hybrid region is very similar. For the runs with an H-width of 30 to 50, the behaviour of the AdResS density distribution in the Full Particle region shares higher similarities with the reference. The amplitude and intersection point are approximately the same. In the Hybrid and Coarse Grain regions, the accuracy is lower. Especially when the resolution changes, such as from FP to H and H to CG, there is a steep change in the density despite the usage of a weight function, which should create a soft transition from one region to another. This is prominent in the tall spike on the right of the fuel's Hybrid region.

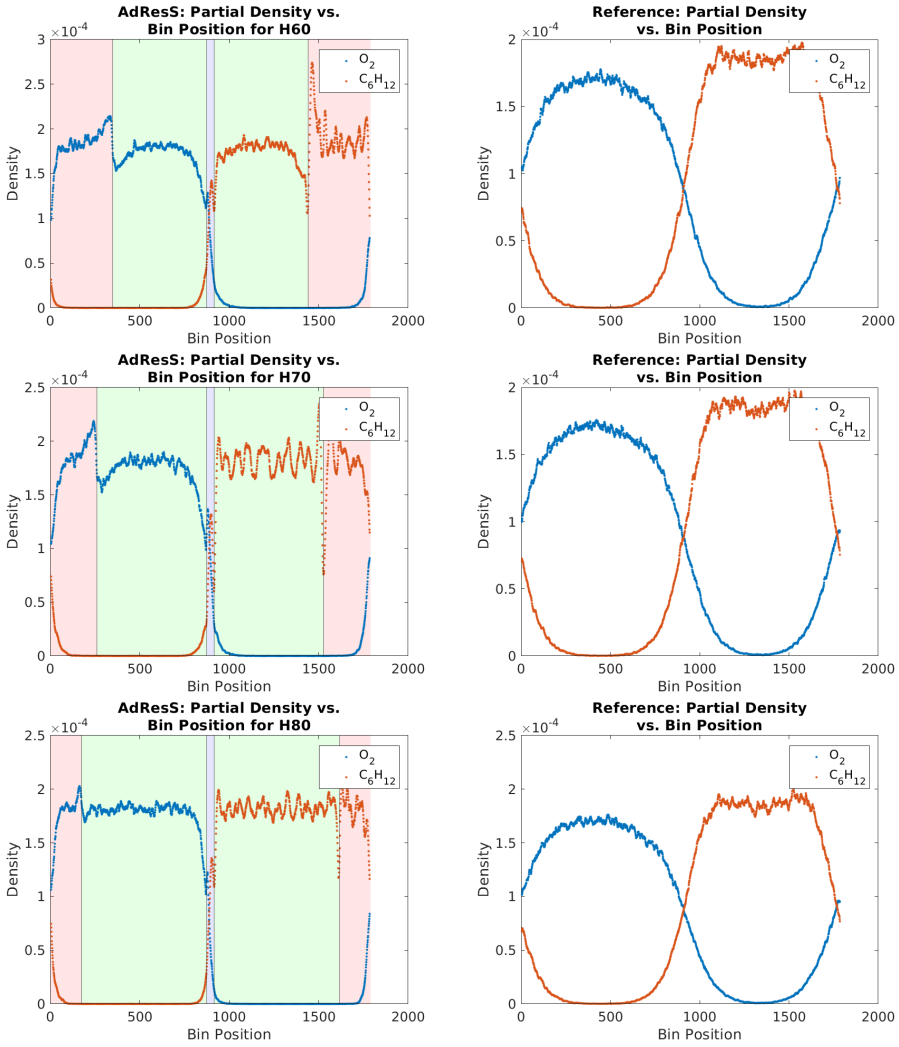


Figure 8.13.: Density distribution of HYBRID C_6H_{12} runs with 60% to 80% Hybrid width (ordered top to bottom). Left column: AdResS, Right column: reference. Background: Red = Coarse Grain, Green = Hybrid, Blue = Full Particle

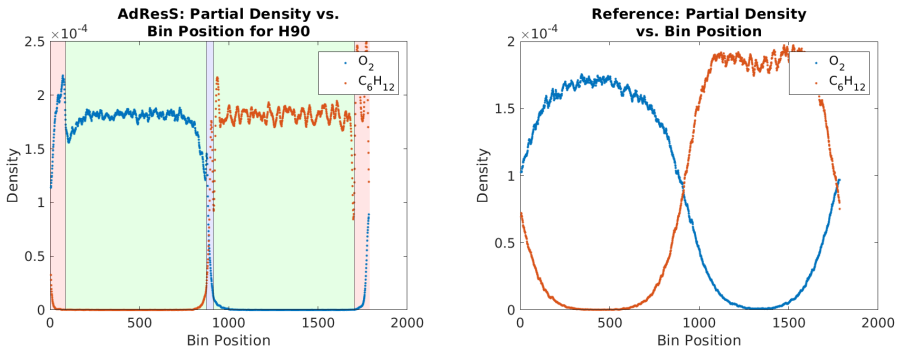


Figure 8.14.: Density distribution of HYBRID C_6H_{12} runs with 90% Hybrid width. Left column: AdResS, Right column: reference. Background: Red = Coarse Grain, Green = Hybrid, Blue = Full Particle

Starting with a H-width of 50%, there are peaks in the interface point in the middle of the domain. These become stronger for higher molecule counts. Up to an H-width of 60%, the density distribution for AdResS in the Full Particle and Hybrid region are rounded and follow the reference. However, after that, the graph of AdResS becomes more rectangular and has a "spiky" profile on the right half.
Based on these findings, we conclude that there is an optimal width for the Hybrid region in terms of accuracy. This presumably depends on the used molecules and must therefore be first found experimentally. In our simulation, this optimum is at a Hybrid width of 40% when we consider the similarity of AdResS to reference in the Full Particle and Hybrid region.
Overall combining the results of both discussed scenarios we derive that our approach is viable when an appropriate width for the Hybrid region is used and, secondly, the application of AdResS only changes small molecules or functional groups of larger molecules.

8.3. Performance

As the first point, we must mention, for clarification purposes, that all AdResS simulations used our implementation based on the `LegacyCellProcessor`, which does not use vectorization. In contrast, all reference simulations were run using the `VectorizedCellProcessor`, which uses SIMD. In the following, we analyse the speedup of our implementation while changing different parameters. Here we check the results of all scenarios (BASE, HYBRID and XSCALE). We ran the benchmarks on two supercomputers to see how our implementation performs on different hardware.
The first system is HSUper, located at HSU in Hamburg. It is based on the Intel Icelake architecture and has up to 576 CPU oriented compute nodes. Each node has 256 GB of RAM and two sockets, each containing an Intel® Xeon® Platinum 8360Y processor with 36 cores or 72 cores per node.[1]
The other system is CoolMUC-2, located at LRZ in Munich. Its architecture is older and is based on Intel Haswell. It has in total 812 nodes with 64 GB of RAM each. Every node has 2 Intel® Xeon® E5-2697v3 processors with 14 cores or 28 cores per node.[2]

8.3.1. Results on HSUper

First, we analyse the strong scaling performance of the C_6H_{12}-BASE scenario. Initially, we ran tests for C_6H_{12} with molecule counts ranging from 4.0e+3 to 1.0e+6. The strong scaling graphs are displayed in figures 8.15 and 8.16. As these did not saturate the computational limits on HSUper, we added runs with 1.0e+7 and 1.0e+8 molecules. In the reference runs, we can observe that for 4.0e+3 and 1.6e+4 molecules, scaling beyond one node retrieves diminishing returns. This is due to the inadequate workload being distributed on many processes, which increases the overhead. The graphs for 6.4e+4, 2.56e+5 and 1.0e+6 molecules rise linearly initially and then reach a plateau. In the beginning, the linear rise is likely due to the big enough workload, such that distributing it on more processes fully outweighs any overhead. Worth noting is that the performance does not drop significantly when scaling far beyond the point, after which no further performance gain was achieved. When the plateau is reached, the overhead dominates. We assume the speedup does not drop for the node counts we used because the nodes are fast enough to compensate for the extra overhead. However, it is probable that with higher node counts, the speed also drops. The runs with 1.0e+7 and 1.0e+8 molecules scale approximately linearly. The linear scaling should also end with higher node counts, but we did not run these tests. The highest reached speedup was with 1.0e+8 molecules on 128 nodes with a factor of about 100. Additionally, the achieved speedups were higher when the simulation contained more molecules because each node had more work, which could run in parallel. The usage of AdResS did not change this behaviour. Only the gaps between the graphs are slightly larger. We attribute these minor changes to the change in the molecular model.

[1] https://www.hsu-hh.de/hpc/en/hsuper/
[2] https://doku.lrz.de/coolmuc-2-11484376.html

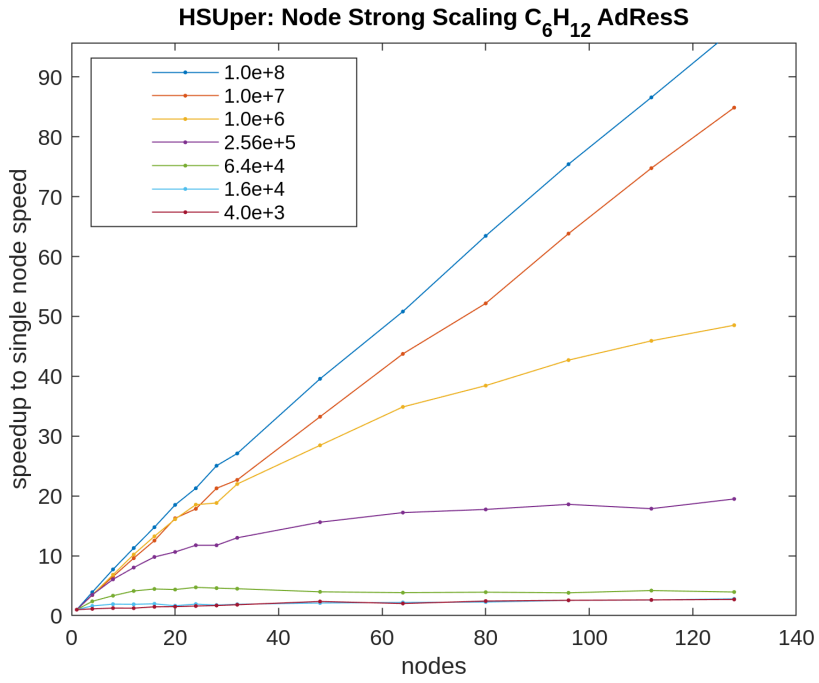


Figure 8.15.: Performance on HSUper: BASE scenario for C_6H_{12} with 4.0e+3 to 1.0e+8 molecules on 1 to 128 nodes using AdResS

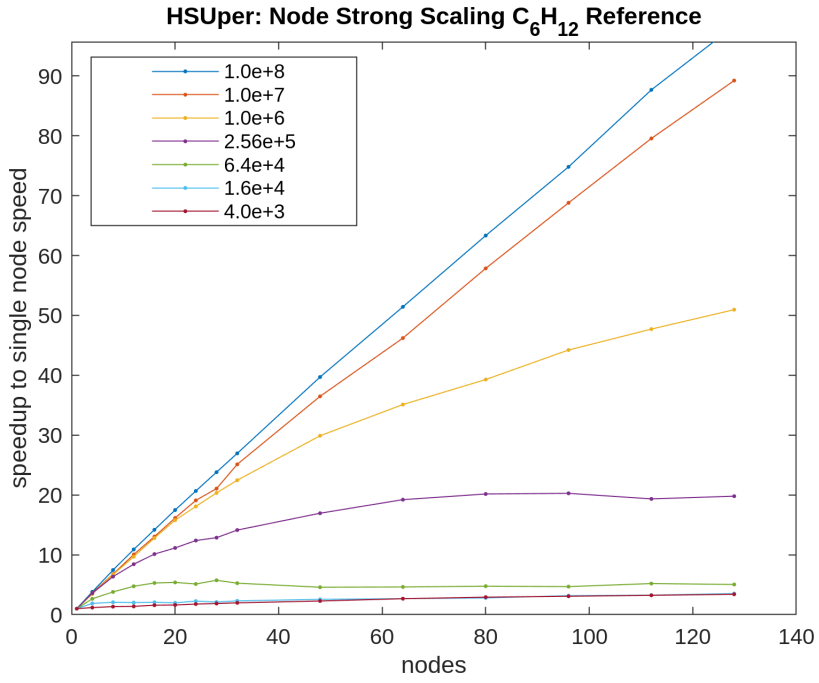


Figure 8.16.: Performance on HSUper: BASE scenario for C_6H_{12} with 4.0e+3 to 1.0e+8 molecules on 1 to 128 nodes using reference implementation.

When comparing the MUPS of using AdResS to the reference, runs with lower molecule counts see a drop in performance, such as with 4.0e+3 and 1.6e+4 molecules. In this case, the introduction of AdResS increased the overhead. In combination with the reduction of total needed force computations, this led to worse strong scaling behaviour. All other runs achieved a maximum speedup of 1.4. As mentioned, we added the simulations with 1.0e+7 and 1.0e+8 molecules because the computational limits were not saturated. This, in turn, was done because we saw higher speedup on our private machines. However, as observable in figure 8.17, this had no impact on HSUper. We assume that the overall small speedup is because the reference used the AVX512 instruction set and thus had a higher baseline performance.

The simulations with CH_4 have similar strong scaling properties as the C_6H_{12} runs because the number of molecules did not change, which is the primary influences on the amount of work. However, the speedup of AdResS compared to the reference has its maximum at 1.2 but is, in most cases, around or less than one due to the reduction of total site-site interactions. This is shown in figures 8.18 and 8.19.

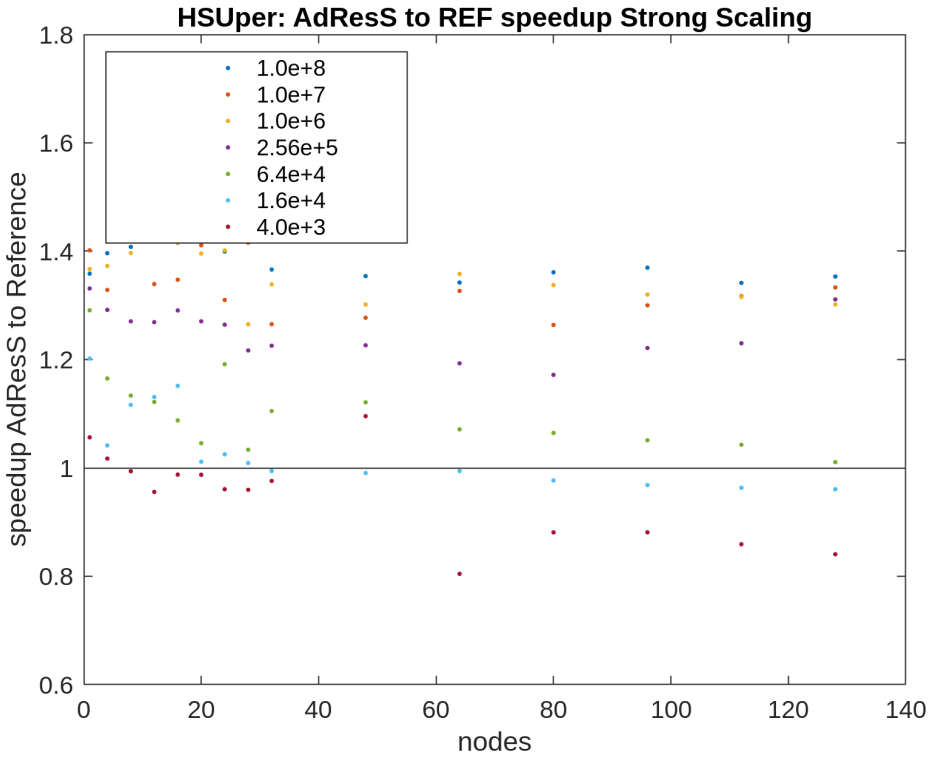


Figure 8.17.: Speedup achieved using AdResS compared to Reference in BASE scenario using 4.0e+3 to 1.0e+8 molecules for C_6H_{12}

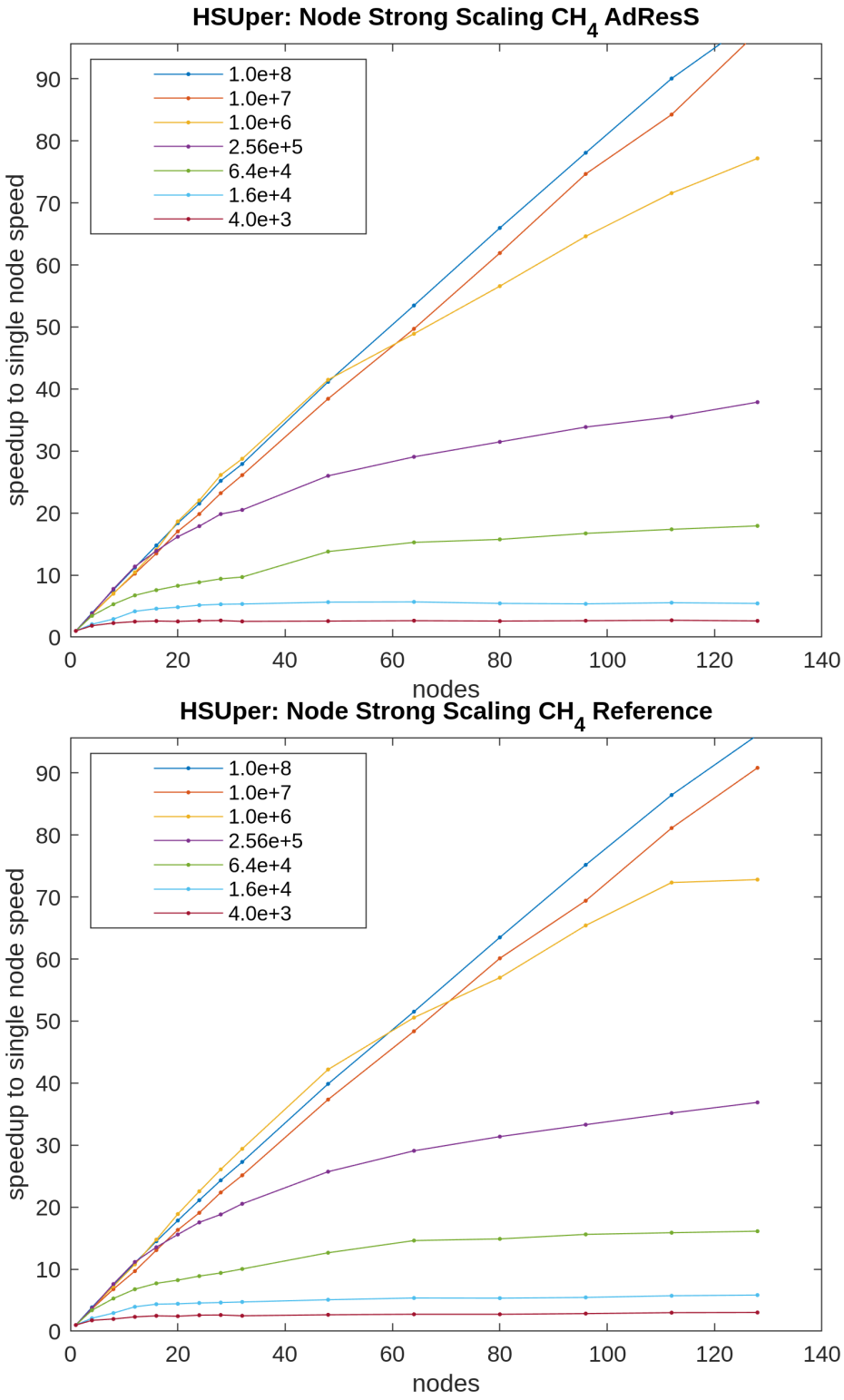


Figure 8.18.: Performance on HSUper: BASE scenario for CH_4 with 4.0e+3 to 1.0e+8 molecules on 1 to 128 nodes. Top: AdResS Bottom: reference)

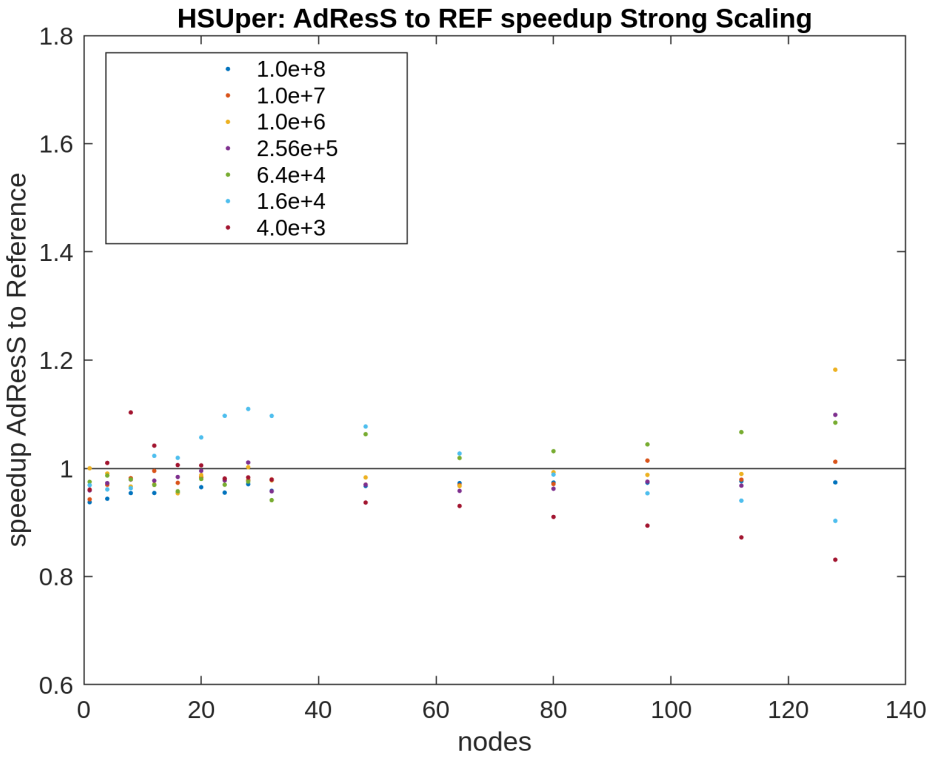


Figure 8.19.: Speedup achieved using AdResS compared to Reference in BASE scenario using 4.0e+3 to 1.0e+8 molecules with CH_4.

To wrap up the finding of our BASE scenario, we conclude that in terms of performance, this scenario was not optimal and that our AdResS approach did not yield significant improvements.

Next, we analyse the performance of the HYBRID scenario. We created this scenario to examine if the width of the Hybrid region has an impact on accuracy. According to our findings, this is indeed the case. Therefore, we also analyse the performance here to see how high the tradeoff between accuracy and speed is. According to the speed data of the C_6H_{12} and CH_4 run, which is displayed in figure 8.20, the performance impact follows expectations. The smaller the Hybrid region, the higher the performance because the CG region is larger. This leads to fewer site-site interactions and, therefore, fewer computations are needed. However, further increments do not affect performance much more once a specific width is reached. The difference between C_6H_{12} and CH_4 is that the gaps between the different Hybrid sizes are smaller, likely due to less total workload for CH_4.

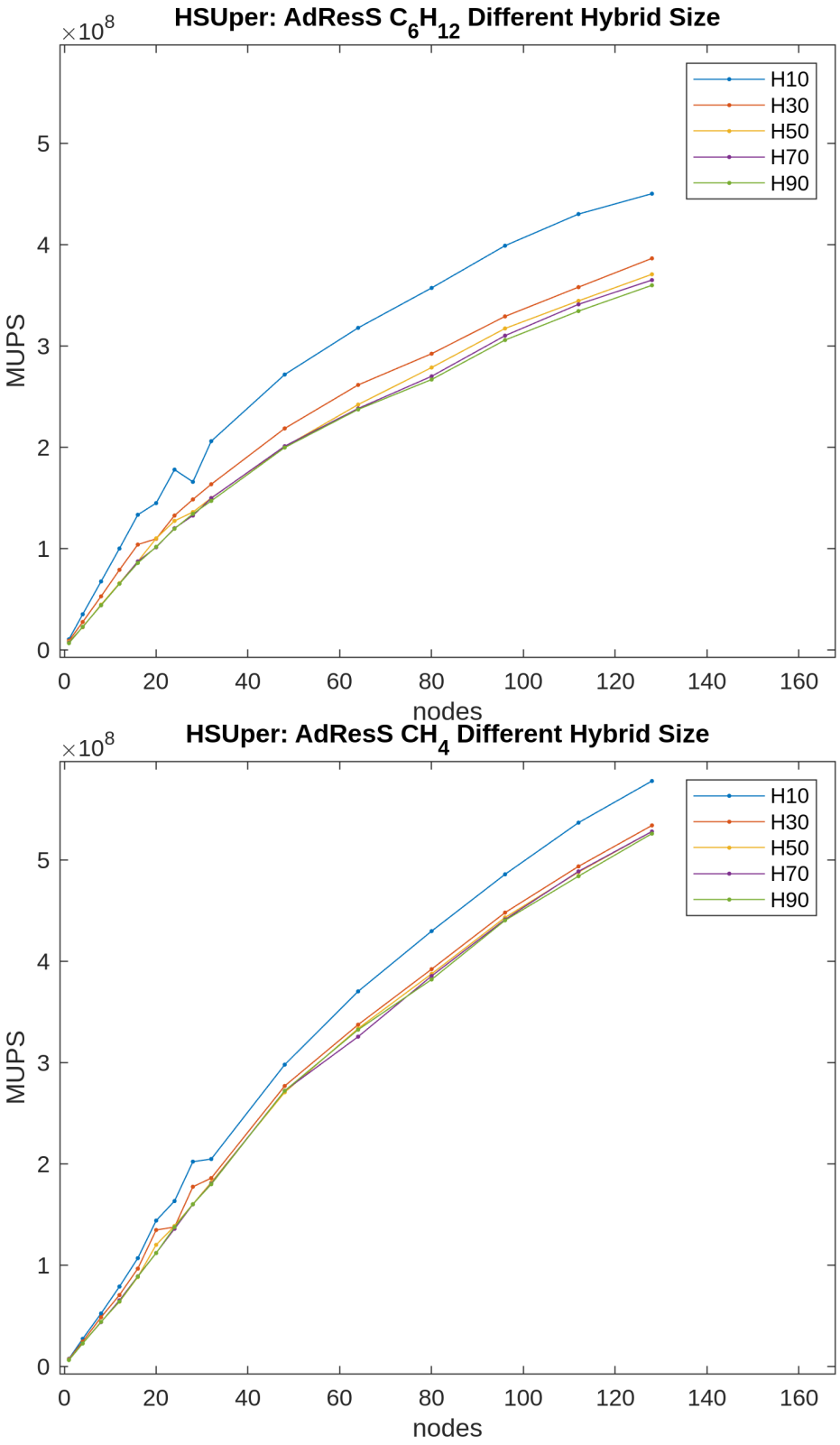


Figure 8.20.: MUPS achieved of using AdResS in the HYBRID scenario for different Hybrid widths ranging from 10% to 90%. Top: C_6H_{12} Bottom: CH_4

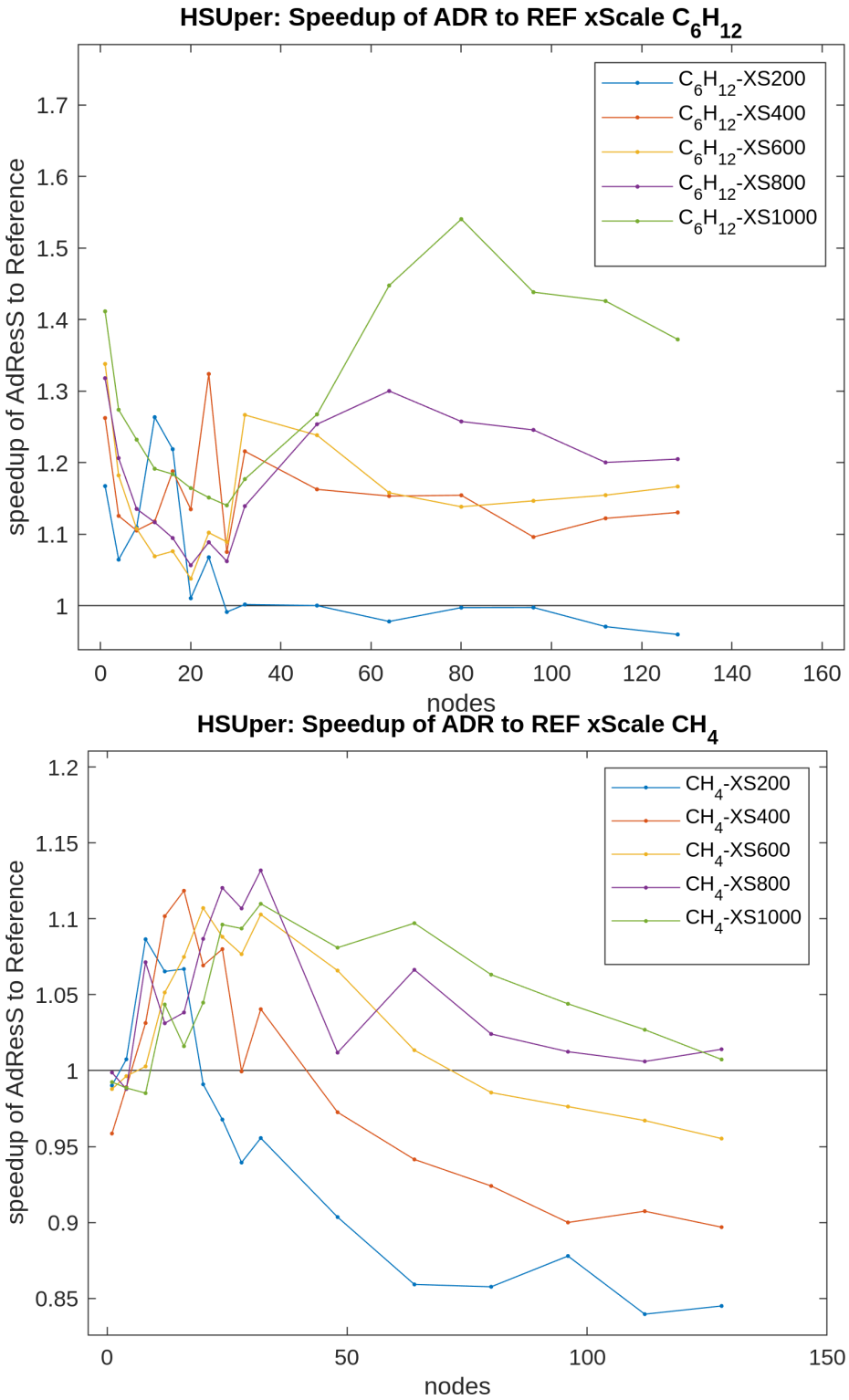


Figure 8.21.: MUPS achieved using AdResS in the XSCALE scenario for different Coarse Grain region sizes ranging from 100 to 1000 units. Top: C_6H_{12} Bottom: CH_4

The remaining scenario is XSCALE. As this represents a more natural use case of AdResS, the resulting speedups may better reflect the actual performance gain. We omit the strong scaling here since it was highlighted in the other simulations in great detail. Therefore, we only check the performance gain using AdResS compared to the reference. Overall, the performance is higher with larger Coarse Grain regions. In the case of C_6H_{12}, the highest achieved speedup was around a factor of 1.55 on 80 nodes. We explain this higher performance by having reduced the amount of work on each node by a higher constant factor. As the force computation's complexity is linear in the number of molecules, this results in good scaling and higher performance. The performance of AdResS drops below the reference only when the CG region is not large enough. In the case of CH_4, the prior speedup of 1.2 was not even reached, and in general, it performed worse. We assume this is the case because adding more CH_4 molecules does not increase the workload enough, as the Coarse Grain and Full Particle representations use the same molecular representation.

8.3.2. Results on CoolMuc 2

In the results for CoolMuc 2, we only present the performance of the BASE simulation using 2.56e+5 and 1.0e+6 molecules because all other aspects are more algorithm dependent, and our intention is not to benchmark the systems, but only the implementation. In the strong scaling graph of the AdResS runs, which are displayed in figure 8.22, the speedups rise approximately linearly, then reach their respective peaks and drop down again. This behaviour is also observable for the reference runs. The linear scaling and peak explanation is analogous to the one for HSUper. The now visible drop in speedup is probably due to CoolMuc 2 being an older system, such that it could not compensate for the extra overhead. However on this system, the reference run achieves the higher maximum speedup of a factor of about 180, whereas the AdResS run only achieved about 130. These highest speedups were both produced by the runs with 1.0e+6 molecules with C_6H_{12} at 48 nodes. The maximum speedup of the AdResS run is lower than the reference because the AdResS run had a higher base performance on a single node.

The comparison of the AdResS performance to the reference in figure 8.23 shows that only the run with 1.0e+6 molecules with C_6H_{12} maintained a speedup of a factor greater than 1. It also had the highest speedup of about factor 2 with a single node. The higher maximum speedup in comparison to HSUper is probably due to CoolMuc 2 only using AVX2. Since this run had the most computational work we assume that AdResS introduced a higher overhead than performance gain for the other runs. The speedups of the CH_4 runs, which are in the beginning all about factor 1, also support this assumption because here, AdResS did not alter the workload a lot. On the other hand, both C_6H_{12} runs have positive speedups for small node counts. Here the workload was reduced by a constant factor.

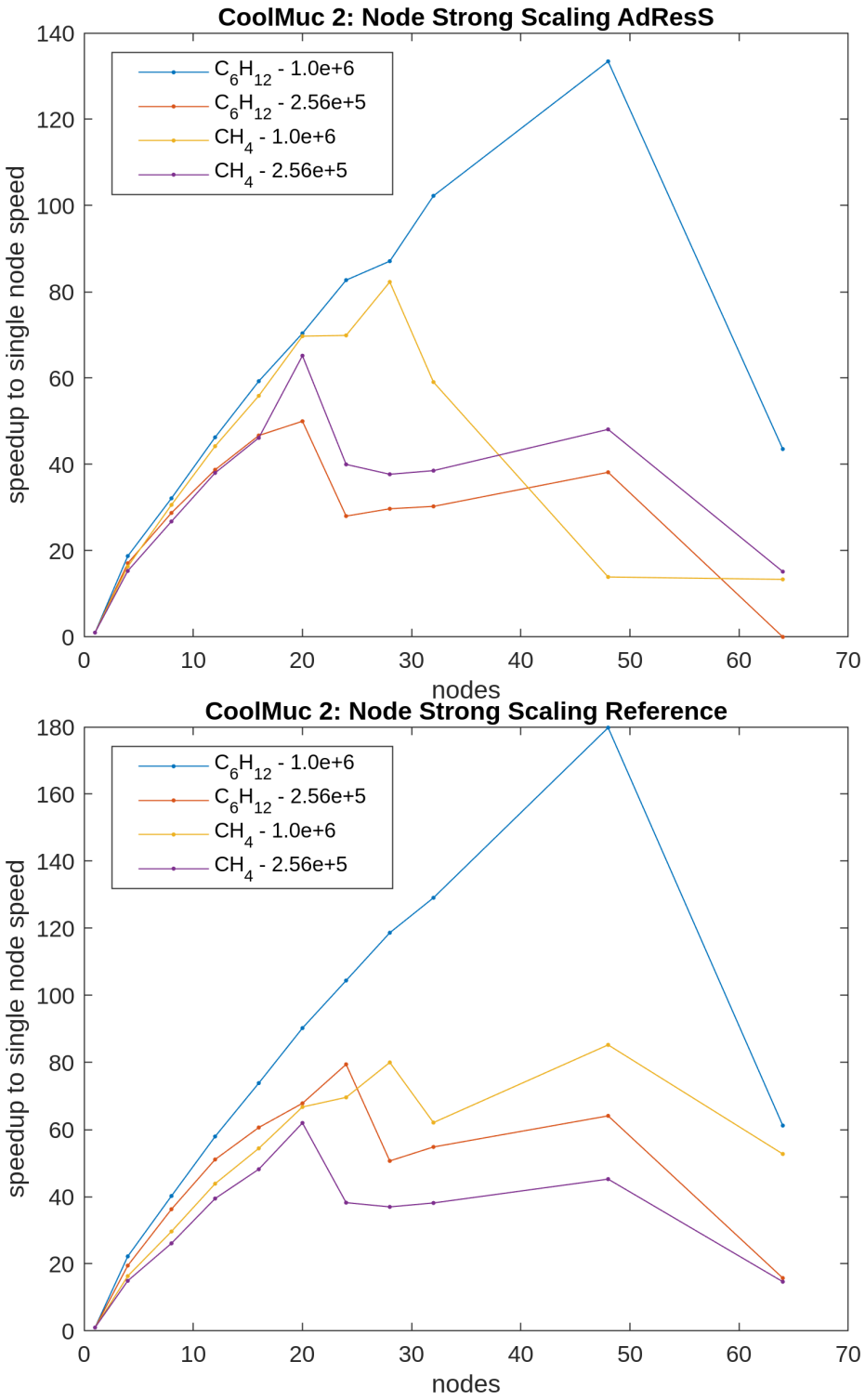


Figure 8.22.: Performance on CoolMuc 2: BASE scenario for CH_4 and C_6H_{12} with 2.56e+5 and 1.0e+6 molecules on 1 to 64 nodes. Top: AdResS Bottom: reference)

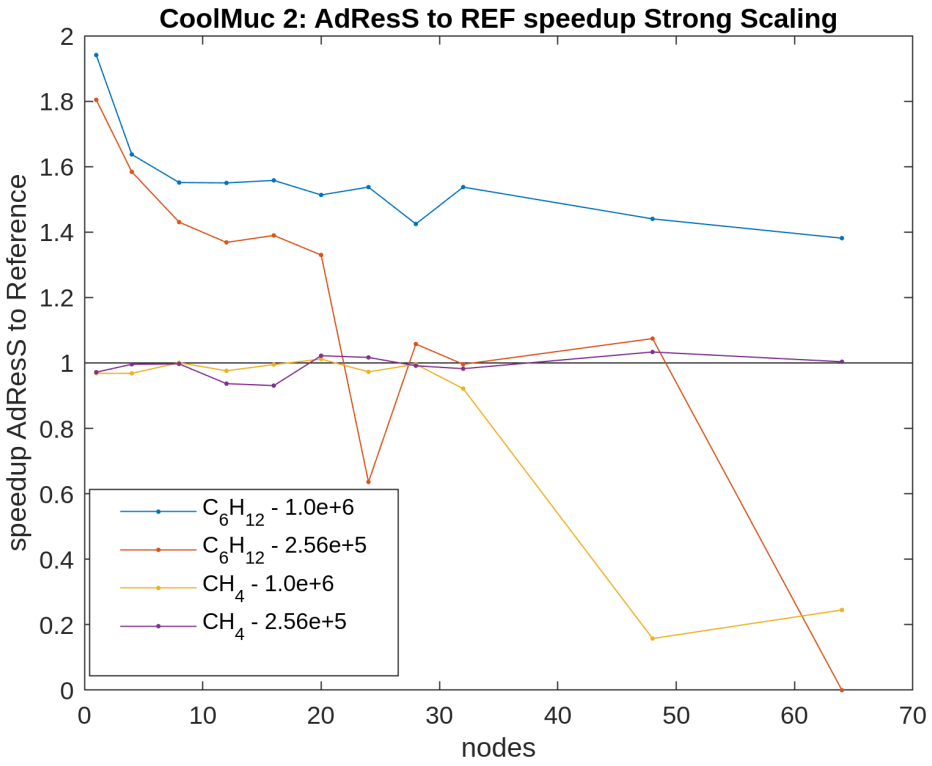


Figure 8.23.: Speedup achieved using AdResS compared to Reference in BASE scenario using 2.56e+5 and 1.0e+6 molecules with CH_4 and C_6H_{12}.

8.3.3. Conclusion of results

Taking all findings into consideration, we conclude that the following aspects should be considered when using this kind of AdResS:

1. The Hybrid width influences accuracy and performance. But accuracy should be the higher priority since the wrong choice of Hybrid width leads to unusable results in most cases.
2. Changing the representation of small molecules does not yield significant performance improvements. However, drastically altering the model of large molecules leads to lower accuracy. Therefore, exchanging functional groups of larger molecules may be the better option.
3. The total achievable maximum speedup depends on the underlying system architecture. Older systems may profit more from using AdResS.

9. Summary and Conclusion

This thesis described a possible alternative to the common AdResS approach. Instead of using a single site for the CG representation, which requires generating a new potential function through RDF sampling, we proposed reusing the same potential functions from the FP representation. The Coarse Graining process also allows for multi-site representations. To find approximately matching molecular representations for the FP and CG model, we queried the MolMod database [15], which contains experimentally generated molecular models. Additionally, we created different weight functions and checked which was the most accurate. Our tests found our `Nearest` approach, which treats the Hybrid region as a box with rounded corners and edges, to be the best option.

Afterwards, we implemented this in an existing MD-platform, ls1-Mardyn, and described our software and the inherent design choices. Due to the complexity of ls1-Mardyn, we opted for the solution which required the least rewriting effort of the methods explored. To determine the accuracy and performance of our AdResS approach, we created different benchmark scenarios and ran them on two supercomputers. The results show that the accuracy increases when the simulation contains more molecules. Furthermore, the impact on accuracy seems to be inversely scaling with the molecule size. The apparent problem is that the molecular representations do not match fully, which leads to perturbed densities in the transition regions between different resolutions. The Hybrid region size also affects accuracy. An appropriate choice can minimise the error in the FP region.
The strong scaling behaviour is not significantly affected by AdResS. We observed that the performance gain was higher when the simulation contained more and larger molecules. Larger Hybrid region sizes reduce the simulation speed, but once a certain size is reached, the growth in speed reduction is limited.

Overall, we conclude that this AdResS approach can be a viable strategy, but it has its limitations. Due to the trade-off between accuracy and performance, we suggest only changing the representation of functional groups of large molecules. Additionally, finding the optimal Hybrid region size should not be omitted. Because of this, further work is necessary. The Hybrid size may be generalized, or more empirical data can be generated. As our implementation did not generate high performance gains, this is an opportunity for future work to either create a vectorized version or extend the simulation to use three-body potentials. Using AdResS in the later example would reduce the computational complexity significantly.

Part III.

Appendix

A. Using AdResS in ls1-Mardyn

A.1. Acquiring ls1-Mardyn

The repository is publicly available at `https://github.com/ls1mardyn/ls1-mardyn`. Since most of our input or configuration files are too large, we refer to the repository using relative paths or names.
The commit-id of our latest version is: 1d3a058e70cdfa86ab3b0baf12fc2916a8885535
Our main implementation can be found in the branch "adress-1-particle-container". The other version, which may still contain bugs, is in the branch "adress-3-particle-container".
The repositories main structure is as follows:

1. /src : source code of ls1-Mardyn
2. /examples : many examples ready-to-run
3. /tools/VTKAnalysis : our vtk-file analysis tool

A.2. Input Files

All AdResS-related configuration files are located in /examples/AdResS. The scenario used during the weight function analysis is in /examples/AdResS/WeightComp. All other scenarios are in /examples/AdResS/BlockInterface.

The weight analysis is organised in 6 main configuration files, config0.xml to config6.xml. To run these, the initialisation run in ./init must first be started, and afterwards, the results must be copied back into the /WeightComp folder.

The BASE, HYBRID and XSCALE scenarios are in the /BlockInterface folder. Since all of those contain many different runs, we created bash scripts to automate the benchmarking process.

1. make_inputs.sh : Goes through all subfolders and creates the BASE and HYBRID scenarios configuration files to equilibrate the simulations. It also generates SBATCH commands for systems which use SLURM. However, those need to be adapted to the local system. When called with the CLEAN=1 parameter, those temporary files are deleted.
2. make_final_runs.sh : Creates the final run configuration files for the BASE and HYBRID scenarios in all subfolders. When called with COPY=1 it copies all result files from equilibration into the correct folders. SBATCH commands are also created.
3. make_xs.sh : Creates in the xScale subfolder the configuration files for the XSCALE scenario to equilibrate the simulations. It also generates SBATCH commands for

systems which use SLURM. However, those need to be adapted to the local system. When called with the CLEAN=1 parameter, those temporary files are deleted.

4. make_final_runs_xs.sh : Creates in the xScale subfolder the final run configuration files for the XSCALE scenario. When called with COPY=1 it copies all result files from equilibration into the correct folders. SBATCH commands are also created.
5. sbatch_q.sh : Runs the specified SBATCH commands. Parameters: INIT_BASE=1 starts the equilibration of the BASE and HYBRID scenarios. BENCH=1 starts the final runs of BASE. BENCH_H starts the final runs of HYBRID. INIT_XS=1 starts the equilibration of XSCALE. BENCH_XS=1 starts the final runs of XSCALE.

The files ./BlockInterface/components_C_6H_{12}.xml and ./BlockInterface/components_CH_4.xml contain the component information for the BASE, HYBRID and XSCALE scenarios.

A.3. Compiling ls1-Mardyn

The regular compilation instructions apply in our main implementation (1 Particle Container). In the other version, only one additional definition is passed to the compiler. Therefore, we include this definition regardless to compile both versions equally. To fully reproduce our results and analysis, VTK support is required. Therefore, we describe the full compilation process here.

Required dependencies and programs:

1. gcc or alternative (we tested with gcc11)
2. MPI support - for example mpich
3. cmake - version 3.11 or newer
4. libxerces - only for VTK support

Compilation steps:

1. cd into the root directory of the cloned repository
2. mkdir build && cd build
3. cmake .. -DCMAKE_BUILD_TYPE=Release -DENABLE_MPI=true
   -DENABLE_VTK:BOOL=ON -DOPENMP:BOOL=ON
   -DENABLE_ADRESS:BOOL=ON -DENABLE_UNIT_TESTS:BOOL=OFF
   -DENABLE_ADIOS2:BOOL=OFF
4. make MarDyn -j (N_CPU*1.5)

A.4. Manually Running Scenarios

After successful compilation ls1-Mardyn can now be executed. To run any previously mentioned configuration files, ls1-Mardyn must be called with the input file as the argument: ./MarDyn ./../../examples/config.xml.

List of Figures

2.1. Methods of Force Computation 4

3.1. Molecular Representation of Tetrahedral Molecule 5
3.2. Potential Functions 6
3.3. Weight Function: Overview and Euclid/Manhattan 8
3.4. Weight Function: Component and Nearest 9
3.5. Comparison of Simulations Using Different Weight Functions 11

4.1. ls1-Mardyn Simple Class Diagram 14

5.1. Main Loop Dataflow Diagram 15

8.1. Scenario BASE Overview 23
8.2. Scenario HYBRID Overview 23
8.3. Scenario XSCALE Overview 24
8.4. Accuracy CH4 4k-16k 25
8.5. Accuracy CH4 64k-1024k 26
8.6. Accuracy CH4 delta 27
8.7. Accuracy C6H12 4k-64k 28
8.8. Accuracy C6H12 256k-1024k 29
8.9. Accuracy C6H12 delta 4k-16k 29
8.10. Accuracy C6H12 delta 64k-1024k 30
8.11. Accuracy Hybrid 10 30
8.12. Accuracy Hybrid 20-50 31
8.13. Accuracy Hybrid 60-80 32
8.14. Accuracy Hybrid 90 33
8.15. Performance HSU BASE C6H12 AdResS 35
8.16. Performance HSU BASE C6H12 Reference 35
8.17. Performance HSU BASE C6H12 ADR to REF Comparison 36
8.18. Performance HSU BASE CH4 37
8.19. Performance HSU BASE CH4 ADR to REF Comparison 38
8.20. Performance HSU HYBRID 39
8.21. Performance HSU XSCALE 40
8.22. Performance CM2 BASE 42
8.23. Performance CM2 BASE ADR to REF Comparison 43

List of Tables

3.1. Overview of the sum of relative error in either full domain or Full Particle region for each implementation. The relative error is measured against the reference implementation. 11

Bibliography

[1] *Computational soft matter: from synthetic polymers to proteins. lect: Lecture notes.* No. 23 in NIC series. NIC.

[2] Agarwal, A., and Delle Site, L. Path integral molecular dynamics within the grand canonical-like adaptive resolution technique: Simulation of liquid water. 094102.

[3] Cortes-Huerto, R., Praprotnik, M., Kremer, K., and Delle Site, L. From adaptive resolution to molecular dynamics of open systems. 189.

[4] Delle Site, L. Some fundamental problems for an energy-conserving adaptive-resolution molecular dynamics scheme. 047701.

[5] Gratl, F. A., Seckler, S., Tchipev, N., Bungartz, H.-J., and Neumann, P. AutoPas: Auto-tuning for particle simulations. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 748–757.

[6] Hollingsworth, S. A., and Dror, R. O. Molecular dynamics simulation for all. 1129–1143.

[7] Nitzke, I., Stierle, R., Stephan, S., Pfitzner, M., Gross, J., and Vrabec, J. Phase equilibria and interface properties of hydrocarbon propellant–oxygen mixtures in the transcritical regime. 032117.

[8] Potestio, R., Fritsch, S., Español, P., Delgado-Buscalioni, R., Kremer, K., Everaers, R., and Donadio, D. Hamiltonian adaptive resolution simulation for molecular liquids. 108301.

[9] Praprotnik, M., Cortes-Huerto, R., Potestio, R., and Delle Site, L. Adaptive resolution molecular dynamics technique. In *Handbook of Materials Modeling*, W. Andreoni and S. Yip, Eds. Springer International Publishing, pp. 1443–1457.

[10] Praprotnik, M., Delle Site, L., and Kremer, K. Adaptive resolution molecular-dynamics simulation: Changing the degrees of freedom on the fly. 224106.

[11] Praprotnik, M., Delle Site, L., and Kremer, K. A macromolecule in a solvent: Adaptive resolution molecular dynamics simulation. 134902.

[12] Praprotnik, M., Site, L. D., and Kremer, K. Multiscale simulation of soft matter: From scale bridging to adaptive resolution. 545–571.

[13] Seckler, S., Gratl, F., Heinen, M., Vrabec, J., Bungartz, H.-J., and Neumann, P. AutoPas in ls1 mardyn: Massively parallel particle simulations with node-level auto-tuning. 101296.

[14] Shadrack Jabes, B., Klein, R., and Delle Site, L. Structural locality and early stage of aggregation of micelles in water: An adaptive resolution molecular dynamics study. 1800025.

[15] Stephan, S., Horsch, M. T., Vrabec, J., and Hasse, H. Molmod – an open access database of force fields for molecular simulations of fluids. *Molecular Simulation 45*, 10 (2019), 806–814.

[16] Tchipev, N. P. Algorithmic and implementational optimizations of molecular dynamics simulations for process engineering.

[17] Wang, H., and Agarwal, A. Adaptive resolution simulation in equilibrium and beyond. 2269–2287.

[18] Wang, H., Hartmann, C., Schütte, C., and Delle Site, L. Grand-canonical-like molecular-dynamics simulations by using an adaptive-resolution technique. 011018.