View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by UNT Digital Library

SAND REPORT

SAND2005-0070
Unlimited Release
Printed January 2005

Modeling Local Chemistry in the Presence of Collective Phenomena

N.A. Modine and M.E. Chandross

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Issued by Sandia National Laboratories, operated for the United States

Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

SAND2005-0070
Unlimited Release
Printed January 2005

Modeling Local Chemistry in the Presence of Collective Phenomena

N.A. Modine
Nanostructure and Semiconductor Physics Department

M.E. Chandross
Materials and Process Modeling and Computation Department

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185

Abstract

Confinement within the nanoscale pores of a zeolite strongly modifies the behavior of small molecules. Typical of many such interesting and important problems, realistic modeling of this phenomena requires simultaneously capturing the detailed behavior of chemical bonds and the possibility of collective dynamics occurring in a complex unit cell (672 atoms in the case of Zeolite-4A). Classical simulations alone cannot reliably model the breaking and formation of chemical bonds, while quantum methods alone are incapable of treating the extended length and time scales characteristic of complex dynamics. We have developed a robust and efficient model in which a small region treated with the Kohn-Sham density functional theory is embedded within a larger system represented with classical potentials. This model has been applied in concert with first-principles electronic structure calculations and classical molecular dynamics and Monte Carlo simulations to study the behavior of water, ammonia, the hydroxide ion, and the ammonium ion in Zeolite-4a. Understanding this behavior is important to the predictive modeling of the aging of Zeolite-based desiccants. In particular, we have studied the absorption of these molecules, interactions between water and the ammonium ion, and reactions between the hydroxide ion and the zeolite cage. We have shown that interactions with the extended Zeolite cage strongly modifies these local chemical phenomena, and thereby we have proven out hypothesis that capturing both local chemistry and collective phenomena is essential to realistic modeling of this system. Based on our results, we have been able to identify two possible mechanisms for the aging of Zeolite-based desiccants.

CONTENTS

1.0 Introduction

Many materials science problems are characterized by the interplay of two disparate phenomena: chemical specificity (CS), in which the rearrangement of chemical bonds must be modeled accurately, and collective phenomena (CP), where new behaviors emerge from the interaction of large numbers of atoms. This situation currently presents an impasse to the computational scientist. Quantum mechanical (QM) methods, which can realistically model bond breaking and formation, are incapable of treating the extended length and time scale involved in CP. On the other hand, classical simulation techniques can explore the longer scales involved in CP, but they typically are not accurate or flexible enough to reliably model aspects of CS such as detailed reaction mechanisms or relative rates. Fortunately, in many situations of interest, the CP are mediated by weaker, longer range interactions (elastic or electrostatic) that can be represented accurately with classical potentials, while the processes that determine CS are localized to small regions where QM methods can be applied efficiently. A generally applicable and reliable approach to exploiting this situation requires coupling classical and QM methods in the same simulation so that a QM treatment can be used in the region where changes in chemical bonding are being studied, while classical potentials are used elsewhere. While some system could, in principle, be understood using separate simulation techniques that target different length and time scales, such an approach cannot adequately address processes that inherently couple effects occurring at both QM and classical length scales.

Of particular interest to Sandia, the difficulties encountered in achieving accurate predictions of the aging and reliability of complex materials are ideally suited to such a multiscale modeling technique. For example, predicting the rate and severity of reactive degradation processes requires an understanding of both the chemical reactions involved, as well as the diffusion of the reacting species to the active sites. As a specific example of such a materials science problem that involves the interaction of CS and CP, consider the aging and reliability of zeolite-based desiccants. Zeolites consist of an aluminosilicate framework, which forms an ordered array of cages defining interconnected nanoscale pores. Interspersed within this framework are sodium and potassium counter-ions, and sufficiently small molecules can diffuse easily through the open crystal structure. Zeolites form the active components of the majority of desiccants used in modern weapons systems.

There is good evidence that chemical specificity plays a major role in the aging of zeolite-based desiccants. Recent experiments show that in addition to water, a number of other gas species present in the weapon's internal atmosphere are absorbed into the zeolite. Experiments have been performed only on the desiccants from the W76 and W80 systems, and in both cases significant, unanticipated adsorption phenomena have been identified. In the case of the W80 desiccant, initial stockpile surveillance data identified excessive premature weight gain, causing concern that the desiccant would drive the refurbishment schedule of the Stockpile Life Extension Program (SLEP). An intense experimental effort eventually attributed this weight gain to the adsorption of carbon dioxide, which has no effect on the desiccant's capacity for water. In contrast, surveillance data from the W76 desiccant indicates that there is a potential precipitous drop-off in water capacity with age. The current hypothesis is that the reduced capacity results from ammonia, which is present from the blowing agent used for the foam encapsulents in the W76. Experiments have so far been unsuccessful in attempts to reproduce this effect, which suggests that a slow, activated

chemical process may be responsible for the changes observed in the stockpile. Realistic treatment of such chemically specific, activated processes demands QM modeling techniques.

The existence of collective phenomena within zeolites are also well established. Collective phenomena occur between different counter-ions, between counter-ions and absorbed molecules, and between different absorbed molecules. For example, studies of faujasite, a zeolite related to Zeolite-3A, have shown that rearrangements of the counter-ions typically involves collective motion of 3 to 6 ions. Furthermore, absorbed molecules can promote the migration of counter-ions to new sites, and the binding configuration and energy of absorbed molecules depend strongly on the configuration of the counter-ions. As a zeolite is loaded with multiple absorbed molecules, interactions between these molecules produce remarkable changes in properties such as diffusion coefficients and binding sites. Finally, in certain cases (such as following ion-exchange at a catalytic centers) macroscopic changes in the crystal structure of certain zeolites have been observed. Investigation of any of these effects for Zeolite-3A would involve modeling at least one full unit cell, which contains over 600 atoms, for a significant period of time. These length and time scales are currently well beyond the practical limits of fully QM approaches. Due to the conflicting demands of simultaneously modeling the length scales inherent to CS and CP, a method that is suitable for realistic modeling of aging phenomena in zeolite-based desiccants did not previously exist.

We have developed a hybrid quantum/classical model bridging the length scales involved in CS and CP and applied it to study the materials science of zeolite-based desiccants. Our approach combines purely quantum mechanical calculations, purely classical calculations, and a technique in which a quantum mechanical region is embedded within a classically represented background. We have used this approach to study the absorption behavior of water, ammonia, the hydroxide ion, and the ammonium ion in Zeolite-4a, as well as interactions between some of these species and with the zeolite cage. Collective relaxation of the Zeolite cage in response to the absorbed molecules strongly influences behavior, and thereby we have proven out hypothesis that capturing both local chemistry and collective phenomena is essential to realistic modeling of this system. In the remainder of this report, we will present results from our quantum mechanical, classical, and hybrid quantum/classical calculations, and then discuss two possible mechanisms for the aging of Zeolite-based desiccants motivated by our results.

2.0 Density Functional Theory Results

Zeolite 4A is an alumino-silicate zeolite with cubic unit cell 24.555 Å on a side. (Fig. 1). Each unit cell contains eight α cages or supercages of diameter 11.4 Å and eight β or sodalite cages of diameter 6.6 Å. The openings to the α cages are eight-membered oxygen rings (8MR) that are approximately 5 Å across. The presence of charge-balancing cations (Na/K for 3A, Na for 4A and Na/Ca for 5A) reduces the effective pore size of the opening to 3 Å or 4 Å, depending on the type. The β cages connect to the α through six-membered rings (6MR) that are too small for most molecules to pass through. The β cages are connected to each other by four-membered rings (4MR). All of the 6-oxygen rings and two thirds of the 8-oxygen rings contain one ion. The remaining one third of the 8-oxygen rings contain two ions.

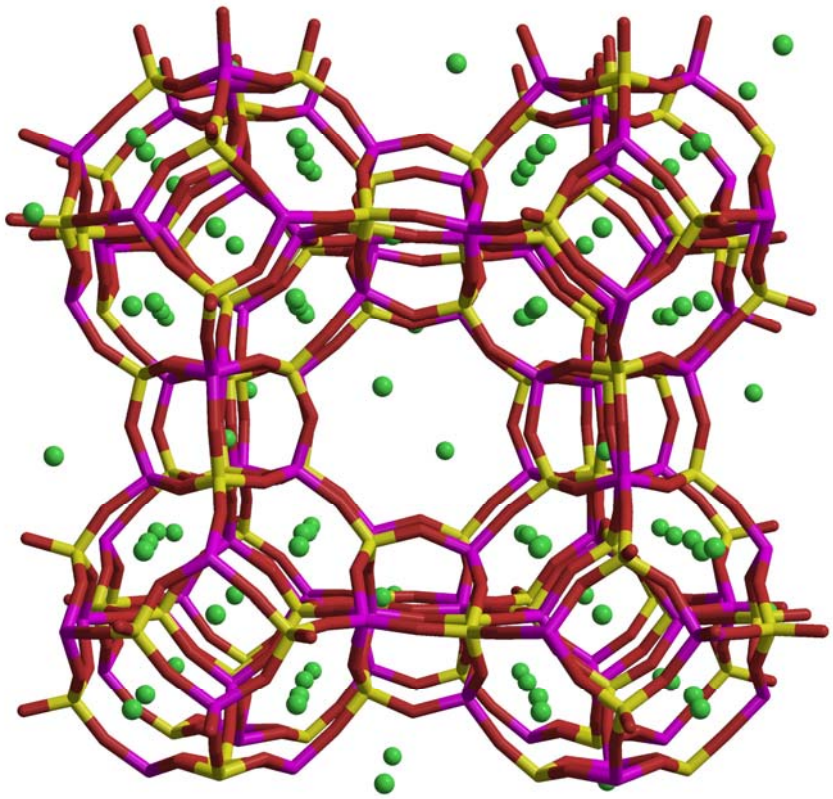

Figure 1. Schematic diagram of zeolite 4A with silicon atoms shown in yellow, aluminum in purple, oxygen in red, and sodium in green.

Although the approach described in this report could have used a variety of quantum mechanical electronic structure techniques, we used the Kohn-Sham density functional theory (DFT). Almost three decades of experience with the DFT has shown that it provides a reasonably accurate representation of bond breaking and formation in a wide variety of systems. Using the DFT, relaxed structures and transition states can be found and short time molecular dynamics can be performed for systems involving up to a few hundred atoms in a practical amount of time. In this project, we applied standard DFT calculations to the relaxation of a variety of different size model systems as well as a few systems involving the full Zeolite-4A unit cell.

2.1 One-Eighth-Cell Model

A full unit cell of Zeolite-4A contains 672 atoms. Applying conventional density functional theory (DFT) methods to such a large structure is extremely time consuming. Therefore, we began our investigation of the materials science of zeolite-based desiccants by studying smaller model system that capture many of the important features of Zeolite-4A. In order to construct our first model system, we periodically repeated one eighth of the Zeolite-4A unit cell. The resulting structure has cage, ring, and ion arrangements that are almost identical to those in the full cell, but the strict Si-O-Al alternation of the Zeolite-4A aluminosilicate cage is interrupted by occasional Si-O-Si or Al-

O-Al bonds. We will refer to our model as our "one-eighth-cell model" of Zeolite-4A. Although not perfect, this model captures many important behaviors of Zeolite-4A, and our preliminary studies using this model were very valuable in directing our future work.

We performed first-principles calculations based on the Kohn-Sham density functional theory for our model system. We confirmed that the VASP and Socorro codes give essentially identical results for this system, and investigated the convergence with respect to a full set of technical parameters. We investigated the convergence with respect to the energy cutoffs used for the wavefunctions and the density, the number of electronic bands included in the calculation, the number of k-points used to sample the Brillouin zone, the hardness of the oxygen pseudopotential, the convergence criteria for the wavefunctions, density, and ionic relaxation, the optimization radii for the real-space pseudopotential projectors, and whether the Na *2p* orbitals are included in valence. We then studied a number of variations on our model system in order to start identifying the types of disorder and collective phenomena that might be present in the Zeolite-4A material. We investigated ion rearrangements by moving a Na ion from a 6-oxygen ring to create an empty 6-oxygen ring and an extra 8-oxygen ring with two Na ions. We also moved a Na from an 8-oxygen ring to create an empty 8-oxygen ring and an extra 8-oxygen ring with two Na ions. These rearrangements cost 1.0 eV and 0.8 eV respectively. These energies are high enough that we would not expect to see a significant population of these modified structures in room-temperature equilibrium. We also investigated modifications to the cage structure by studying the antisite pair created by substituting a Si atom for an Al atom in one region of the material, and substituting an Al atom for a Si atom in another well separated region. In order to preserve local charge neutrality, it was assumed that the Na ion moves with the Al atom creating an extra 8-oxygen ring with two Na ions. This antisite pair cost 0.9 eV, and these defects should be very rare in room-temperature equilibrium. Although it is possible that these defects could be frozen into the material during the growth process or results from deviations from ideal stoichiometry, experimental investigations have shown that such defects are indeed very rare.

Next, we investigated the absorption of a single water molecule in our model structure. Starting from 27 different initial configurations for the H_2O molecule inside the large pore of the Zeolite, we relaxed the structure to find 14 different (not related by symmetry) locally stable structures. Four of these structures were within 0.1 eV of the lowest energy structure, and should occur in significant proportions in room temperature equilibrium.

The structures shown in Fig. 2 are characteristic of the low energy structures that we found for a single water molecule. The standard understanding of absorption in zeolites, based on previously studied molecules such as hydrocarbons, O_2, and N_2, is that the barrier for diffusion occurs when the molecule moves through the 8-oxygen rings. In remarkable contrast, we find that all the low energy configurations for water have the molecule located within the 8-oxygen rings. We believe that this results from the polar and hydrogen bonding character of H_2O, and indicates that molecules with these characteristics are likely to behave in a new and different manner.

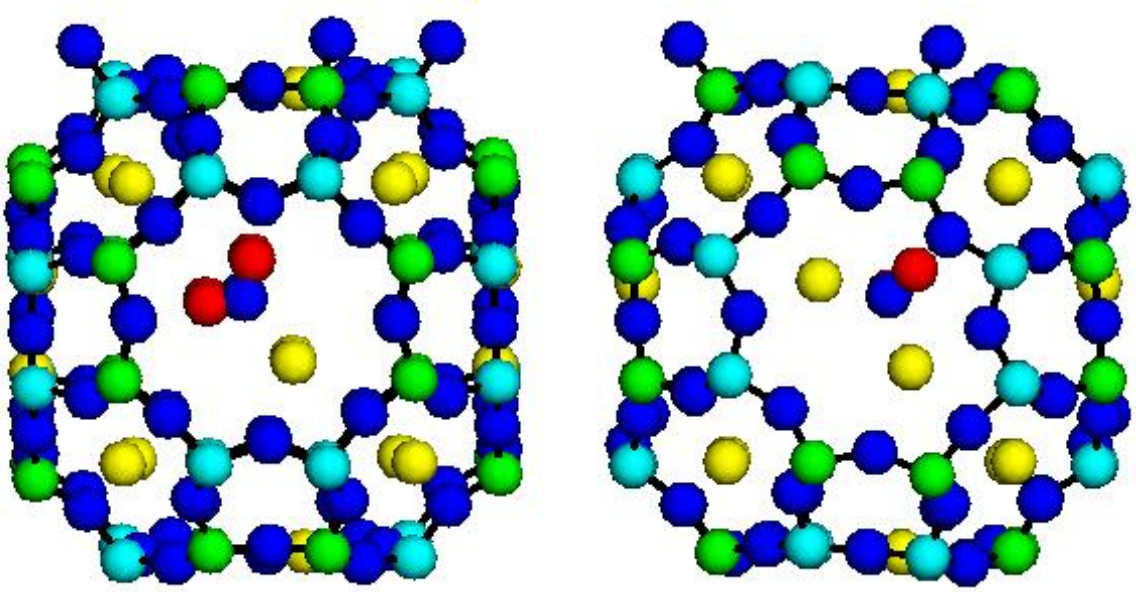

Figure 2. Two low energy structures for a water molecule absorbed in our model structure. The dark blue balls indicate O atoms, the light blue balls indicate Al atoms, the green balls indicate Si atoms, the yellow balls indicate Na atoms, and the red balls indicate H atoms. In (a), the water molecule is located in an 8-oxygen ring containing a single Na ion, while in (b) the water is located in an 8-oxygen ring containing two Na ions.

The first structure in Fig. 2 shows the lowest energy H_2O absorption structure that we initially found for the one-eighth-cell model. The water lies in the plane of the 8-oxygen ring with its O atom oriented toward the Na ion, and its H atoms each forming a hydrogen bond to a ring oxygen. In our highly ordered model system, there are two versions of this structure depending on whether the Si or Al atoms of the 8-oxygen ring containing the H_2O are closest to the rings with two Na ions. The structure where the Al atoms are closest is 0.04 eV higher in energy, and we can probably take this as typical of the strength of inter-ring interactions. The second structure in Fig. 2 shows H_2O absorbed in a 8-oxygen ring containing two Na ions, and for the one-eighth-cell model, its energy is 0.1 eV higher than the first structure. The molecule turns perpendicular to the plane of the 8-oxygen ring. Its O atom is able to interact with both Na ions, but only one of its H atoms is able to form a hydrogen bond. We believe that this perpendicular configuration reduces the area needed by the H_2O molecule, and hence, crowding within the 8-oxygen ring. There is also an asymmetric variant of the two Na ion structure in which both Na ions are displaced to one side of the 8-oxygen ring. The H_2O remains in a perpendicular configuration, but this further reduces crowding, and its energy is intermediate between the two structures in Fig. 2. Despite substantial efforts, we were unable to identify any structure with water absorbed in a 6-oxygen ring. It is likely that H_2O will not fit in these smaller rings. The strong crowding effects apparent with a single H_2O molecule suggest that it will not be thermodynamically feasible to put more than a couple H_2O molecules in each 8-oxygen ring. However, the Na ions may move out of the rings at higher H_2O loading, opening up space and leading to highly collective behavior between the H_2O molecules and the ions.

2.2 One-Half-Cell Model

Although the one-eighth-cell model discussed above has cage, ring, and ion arrangements that are almost identical to those in the full cell, the small size of the cell limits the amount of relaxation that can occur as the zeolite cage accommodates absorbed molecules. Since our results from the one-eighth-cell model indicated that crowding within the 8-oxygen rings was an important consideration, this failure to properly include collective relaxation of the zeolite cage was a cause for concern. Furthermore, in the one-eighth-cell model, the strict Si-O-Al alternation of the Zeolite-4A aluminosilicate cage is interrupted by occasional Si-O-Si or Al-O-Al bonds. Our results showing that small molecules are absorbed within the 8-oxygen rings suggested that a more accurate model of molecular adsorption could be obtained by periodically repeating one-half of the Zeolite-4A unit cell. This one-half cell model gives much greater freedom for the zeolite cage to respond to absorbed molecules while also restoring the strict Si-O-Al alternation of the 8-oxygen rings in one plane. For a molecule absorbed into one of these 8-oxygen rings, the structure of this one-half cell model is identical to the full Zeolite-4A structure out to the seventh neighbor. Nevertheless, computations for this one-half-cell model are still much less computationally expensive than calculations for a full unit cell of the zeolite. After relaxing the structure, we studied the energetic cost of moving a Na from an 8-oxygen ring to create an empty 8-oxygen ring and an extra 8-oxygen ring with two Na ions. This rearrangement costs 0.7 eV, which is in good agreement with the 0.8 eV obtained from the one-eighth cell model.

Next, we used the one-half-cell model to study the low energy configurations of absorbed water identified during the one-eighth-cell calculations. In contrast to our one-eighth-cell results, we found that a H_2O molecule energetically prefers an 8-oxygen ring containing 2 Na ions (shown in Fig. 3) by 0.5 eV over one containing a single ion. Furthermore, the water molecule lies in the plane of the 8-oxygen ring even when located in a window containing two ions. These results can be easily understood in terms of the collective relaxation of the atoms of the zeolite framework in response to the absorbed molecule. In the one-half-cell model, there is sufficient room for the atoms to move outward from the absorption site reducing crowding within the 8-oxygen ring. The resulting increase in area within the 8-oxygen ring allows the water molecule to turn parallel to plane of the ring and to form a second hydrogen bond to the ring oxygens. This lowers the energy of absorption in the two-ion site relative to the more constrained one-eighth-cell model where the collective relaxation of the cage is artificially inhibited by periodic boundary conditions. To confirm that this difference between the one-eighth-cell and one-half-cell results arises mainly from the inhibition of collective relaxation rather than from the interruption of the Si-O-Al alternation in the one-eighth-cell, we considered an alternative version of the one-eighth-cell where the Si-O-Al alternation around the 8-oxygen rings is preserved. This lowered the energy of the two ion absorption site until it was 0.1 eV lower than the one ion site, but this is still much less than the 0.5 eV difference obtained from the one-half-cell model. Finally, we were inspired by our one-half-cell results to see if we could force the one-eighth-cell model into a configuration where the water molecule lies in the plane of a two ion window. We succeeded in creating such a configuration with an energy 84 meV lower than the perpendicular structure for the two ion absorption site. The energy of this structure is very close to the energy of the one ion absorption site (15 meV higher), but this is still clearly a very different result than obtained from the one-half-cell model. In this structure, the Na ions are displaced to one side of the 8-oxygen ring, and we failed to find this configuration in our original search due to a energetic barrier associated with this displacement.

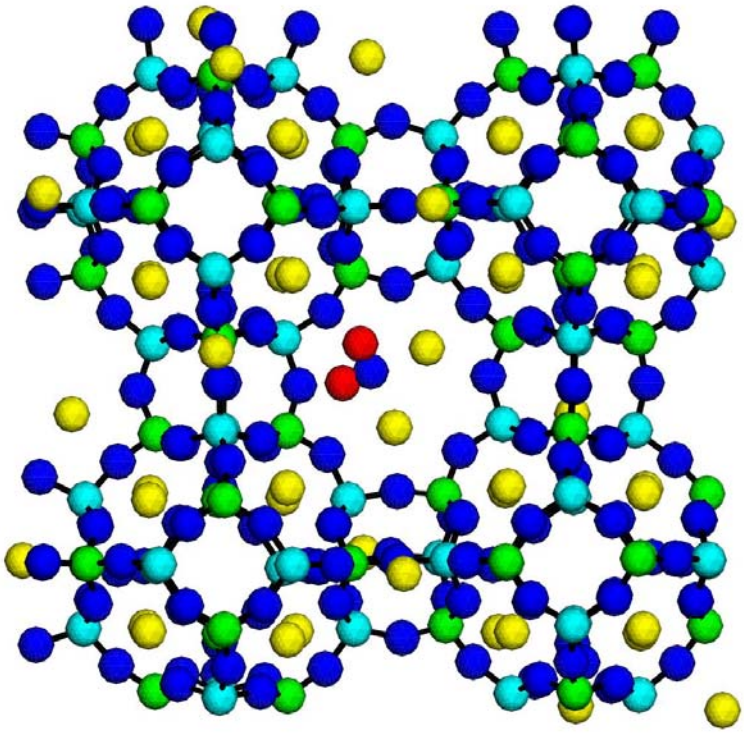

Figure 3. Model of Zeolite-4a that is one half the size of the full unit cell with a water molecule in a 2 ion window. The dark blue balls indicate O atoms, the light blue balls indicate Al atoms, the green balls indicate Si atoms, the yellow balls indicate Na atoms, and the red atoms indicate hydrogen atoms. Notice the uninterrupted Si-O-Al alternation around the 8-membered rings.

2.3 Full Cell Calculations

Given the conflicting results between our one-eighth cell model and one-half-cell model calculations, we decided that we should perform a limited number of calculations using the full unit cell of Zeolite-4A. These calculations provide a reference point for evaluating the accuracy of our model cells as well as our hybrid quantum/classical model. We relaxed the base unit cell of Zeolite-4A and unit cells with a water molecule absorbed into one and two ion sites. Our results are shown in Fig. 4. Notice that one of the Na ions in the window containing the water molecule moves to the side to allow more room for the water. The third ion that appears to be very close to the water is in a window in the other half of the unit cell. Our full-cell DFT calculations indicated that the water prefers the two ion absorption site over the one ion site by 0.45 eV, in good agreement with the half-cell-model.

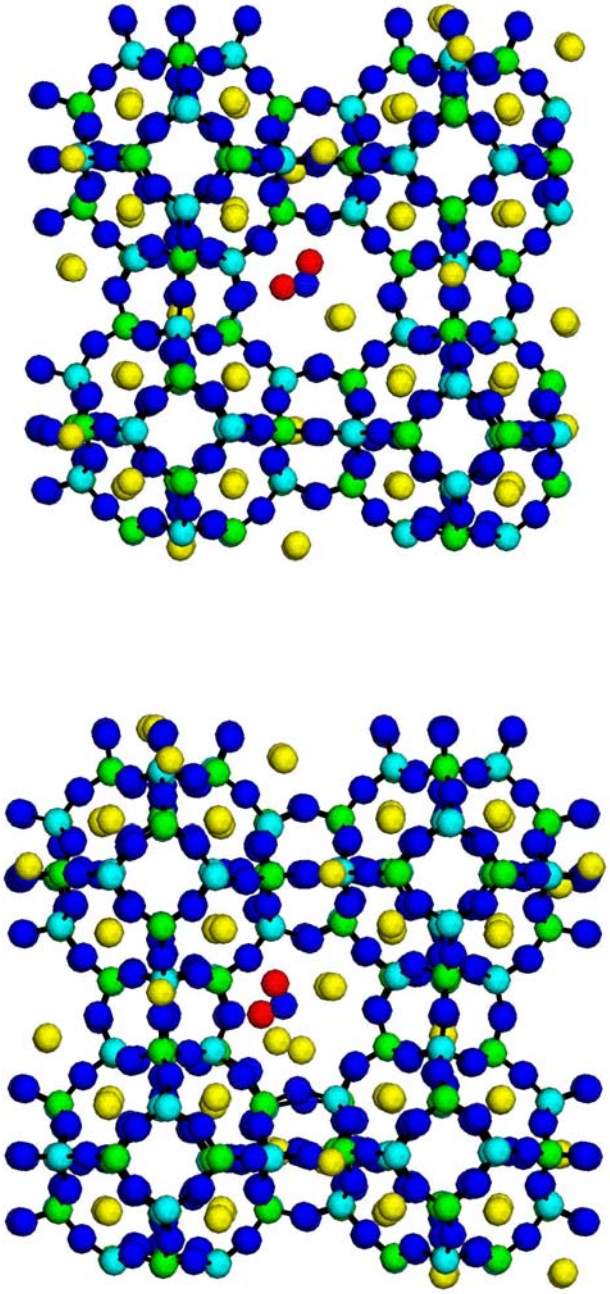

Figure 4. Results of DFT calculations for a full unit cell of Zeolite-4A with a water molecule absorbed into (a) a 1 ion window and (b) into a 2 ion window. The dark blue balls indicate O atoms, the light blue balls indicate Al atoms, the green balls indicate Si atoms, the yellow balls indicate Na atoms, and the red balls indicate hydrogen atoms.

3.0 Classical Force-Field Results

In this project, we depend on classical force-fields to handle the collective behavior of the zeolite and the absorbed molecules. Therefore, it is necessary to make sure that the classical force-fields are accurately capturing phenomena such as loading of the zeolite and competitive absorption. To this end, we focused our classical simulations on developing and verifying force fields for NH3, H2O, CO2 and zeolite-4A. The results of these simulations are compared to experimental results and used to refine our classical representation of the system.

3.1 Introduction

Other groups have reported theoretical and computational studies on water adsorption in zeolite 4A. Lee *et al.*[1] and Faux and co-workers[2-4] have performed MD in hydrated zeolite 4A with fixed numbers of water molecules in order to observe their diffusion and locations in the zeolite cell. Koh and Jhon[5] obtained the positions and differential and integral heats of sorption of water in zeolite 4A using a theoretical approach. In the only paper that reports MD of carbon dioxide in related zeolites, Mizukami *et al.*[6], studied the separation of CO_2 and N_2 using a zeolite NaY membrane. We are not aware of any computational studies of ammonia in zeolite 4A.

In the present paper we report the development of potential parameters for CO_2 and NH_3 which are optimized for adsorption in zeolite 4A. We present adsorption isotherms obtained by Gibbs Ensemble Monte Carlo simulations for CO_2, NH_3 and H_2O at different temperatures, and identify the geometry of the adsorption sites and their dependence on loading. Comparison to available experimental data is also shown.

The remainder of this section is organized as follows: Section II outlines the details of our approach, including the new parameterization and the calculation of adsorption isotherms. Section III provides results and discussion of the force field accuracy, isotherm data and adsorption site geometry, and Section IV offers a summary of our findings as well as concluding remarks.

3.2 Methodology

Here we describe a series of Gibbs Ensemble Monte Carlo (GEMC) simulations of ammonia, carbon dioxide and water in zeolite 4A, to obtain the corresponding adsorption isotherms at various temperatures and to determine the geometry of the adsorption sites. In what follows we describe the models of the zeolite and guest atoms, the potential developed and the GEMC simulations.
A. Zeolite and Guest Molecule Models. Our zeolite model is identical to that used by Faux, *et al*[2-4]. One unit cell of zeolite 4A contains 96 Si, 96 Al, and 384 O atoms bridging the Si and Al. Löwenstein's rule[7] preventing Al-O-Al linkages was enforced by alternating Si and Al tetrahedra. To balance charge, 96 Na atoms are located in the structure as follows, 64 in sites Na(1), 24 in sites Na(2) and 8 in sites Na(3) corresponding to 6MR, 8MR and 4MR windows respectively.[2-4,8-9] The GEMC method allows for the insertion of molecules throughout the zeolite without regard to the physical diffusion pathways. Molecules of both NH_3 and CO_2 are therefore placed within the β-cages, even though this is unphysical. For these molecules, purely repulsive ghost atoms of zero mass and charge were placed at the center of the β-cages to prevent placement inside these cages.

Because water molecules can move through the 6MR window between the supercage and the β-cage, ghost atoms were not used during the calculation of H_2O adsorption.

B. Zeolite and Guest Potentials. The zeolite potential used in this work is that used by Faux and co-workers.[2-4] It contains two-body Coulombic, Lennard-Jones, and Buckingham terms. The form of the potential is:

$$V_{ij} = q_i q_j / r_{ij} + A e^{-r_{ij}/} - C r_{ij}^{-6} + 4 \quad _{ij}[(\quad _{ij}/r_{ij})^{12} - (\quad _{ij}/r_{ij})^{6}]$$

The potential charges used are given in Table 1. The potential parameters employed for the calculation of NH_3 and CO_2 isotherms were fit to the experimental adsorption isotherms at 298 K and are given in Tables 2 and 3 respectively. Two potentials have been used for water isotherms: the SPC/E potential[14] (see Table 4) and the TIP3P potential (see Table 5).[15] The adsorbed molecules were taken as rigid, with bond lengths of 1.010 Å for the N-H bond in NH_3,[16] 1.143 Å for the C-O bond in CO_2,[17] and 1.000 Å and 0.9572 Å for the O-H bond of H_2O for the SPC/E and TIP3P potentials respectively. The bond angles were taken as 106.4°for H-N-H,[16] 180.0° for O-C-O and 109.47° and 104.52° for H-O-H for the SPC/E and TIP3P potentials respectively.

Table 1. Partial charges used in Eq. 1 for the interaction potentials. The subscripts Z, N, C, and W refer to an atom in the zeolite, NH_3, CO_2 and H_20 molecules, respectively.

Atom	Charge (e)
O_Z	-1.86875
Si	3.700
Al	2.775
Na	1.000
N	-1.020
H_N	0.340
C	0.800
O_C	-0.400
O_W (SPC/E)	-0.8476
H_W (SPC/E)	0.4238
O_W (TIP3P)	-0.8340
H_W (TIP3P)	0.4170

Table 2. Potential parameters for NH_3 isotherms.

Species		ε_{ij} (kcal mol^{-1})	σ_{ij} (Å)
O_Z	- N	0.208	3.230
O_Z	- H_N	0.087	2.770
Na	- N	0.082	3.310
Na	- H_N	0.035	2.850
N	- N	0.170	3.420
N	- H_N	0.071	2.960
H_N	- H_N	0.000	0.000

Table 3. Potential parameters for CO_2 isotherms

Species		ε_{ij} (kcal mol^{-1})	σ_{ij} (Å)
O_Z	- C	0.122	2.897
O_Z	- O_C	0.236	3.255
Na	- C	0.048	2.977
Na	- O_C	0.094	3.335
C	- C	0.058	2.753
C	- O_C	0.113	3.112
O_C	- O_C	0.219	3.470

Table 4. SPC/E potential parameters for H_2O isotherms.

Species	ε_{ij} (kcal mol^{-1})	σ_{ij} (Å)
O_Z - O_W	0.560	2.495
Si - O_W	0.186	1.621
Al - O_W	0.118	1.692
O_Z - H_W	0.255	2.227
Si - H_W	0.085	1.354
Al - H_W	0.054	1.425
O_W - O_W	0.155	3.166
O_W - H_W	0.000	0.000
H_W - H_W	0.000	0.000

Species	A_{ij} (kcal mol^{-1})	ρ_{ij} (Å)
Na - O_W	134941.9	0.2387

C. Gibbs Ensemble Monte Carlo Simulations. All simulations were performed using the Monte Carlo for Complex Chemical Systems program (MCCCS Towhee) developed by Martin and co-workers.[12] The isotherms were calculated with the Gibbs Ensemble Monte Carlo method, allowing guest particle moves, box volume changes and guest particle transfers between boxes. The zeolite frame and the extra-framework cations were kept fixed, while guest molecules were taken as rigid bodies.

D. Fitting of Parameters. Several sets of force fields were considered for the simulation of ammonia[16,18,19] and carbon dioxide isotherms.[17,20-26] Of these, some were not optimized for the simulation of zeolite adsorption[16,26] while some were used for zeolites lacking extra-framework cations.[17,21-23] The remaining were used in the simulation of Faujasites[19,20,24,25] and were tested in the current context, but proved not to be transferable or produced distorted isotherms. The parameters used in this work were therefore inspired by parameters from the literature, but primarily obtained by trial and error, fitting them to the available experimental isotherms.

Table 5. TIP3P potential parameters for H_2O isotherms.

Species	ε_{ij} (kcal mol^{-1})	σ_{ij} (Å)
O_Z - O_W	0.555	2.487
Si - O_W	0.184	1.614
Al - O_W	0.117	1.685
O_Z - H_W	0.255	2.227
Si - H_W	0.085	1.354
Al - H_W	0.054	1.425
O_W - O_W	0.152	3.151
O_W - H_W	0.000	0.000
H_W - H_W	0.000	0.000

Species	A_{ij} (kcal mol^{-1})	ρ_{ij} (Å)
Na - O_W	134941.9	0.2387

3.3 Results and Discussion

A. NH_3 Adsorption Isotherms. The potential parameters for NH_3 adsorption were fit to the experimental isotherm at 298 K and then used to calculate the isotherms at 323 K, 343 K and 393 K. Figure 5 shows the adsorption isotherms for NH_3 on zeolite 4A at different temperatures and their comparison with experimental results obtained by Helminen *et al.*[10-11] The errors in the simulations are approximately 10% of the calculated values, and are not shown in the figures. The agreement between the experimental and simulated isotherms is very good and it is remarkable that it extends over five orders of magnitude in pressure and 100 K in temperature. Six different adsorption sites were identified for NH_3 in zeolite 4A. Two are single cation adsorption sites, while in four of them the ammonia molecule is interacting with two Na cations. The two single cation adsorption sites, hereafter A and B, correspond to binding to a Na(1) cation and a Na(2) cation respectively. No ammonia was detected bound to a single Na(3) cation, perhaps because the proximity of the framework atoms makes this arrangement unfavorable. The four two-cation binding sites correspond to binding to Na(1) and Na(2) cations (hereafter C), Na(1) and Na(3) cations (hereafter D), binding to two Na(2) cations (hereafter E) and binding to Na(2) and Na(3) cations (hereafter F). Table 6 shows the average distances between cations and the atoms of ammonia molecules adsorbed in zeolite 4A. Binding sites D and F have the largest distances between the cations and the atoms of ammonia, demonstrating the avoidance of close contacts with framework atoms. The average nitrogen-nitrogen distance in a saturated zeolite 4A is 4.3 Å, which shows the close proximity of the adsorbents in the zeolite.

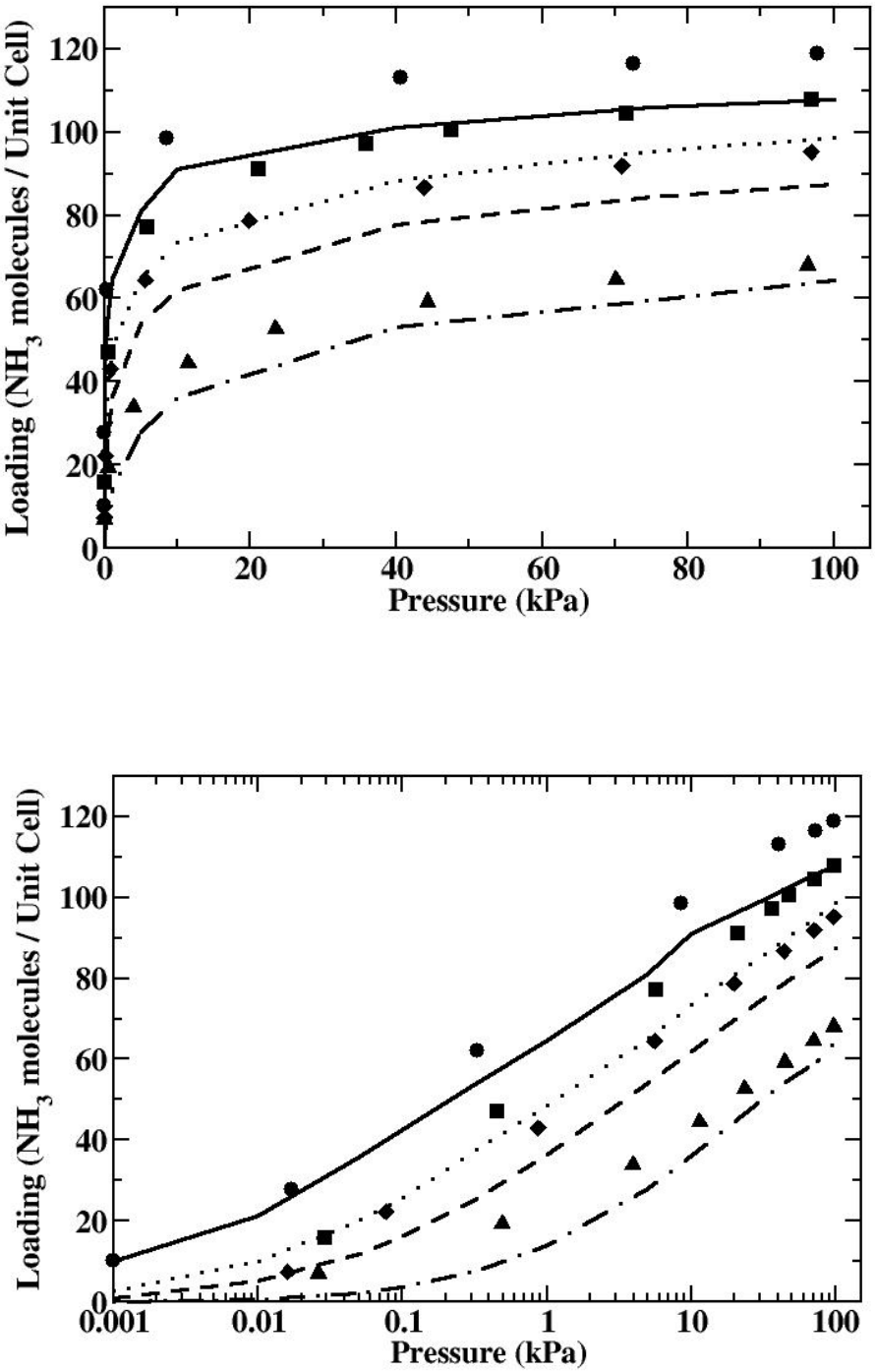

Figure 5. Experimental and simulated adsorption isotherms for NH_3 on zeolite 4A in linear (top) and logarithmic (bottom) scales. Experimental isotherms from Helminen et al.[10-11] at 298 K (●), 323 K (■), 343 K (◆), and 393 K (▲). Simulated isotherms at 298 K (——), 323 K (·····), 343 K (----), and 393 K (·– –).

Table 6. Average distances between cations and atoms of NH_3 adsorbed in zeolite 4A.

Binding Site	Cation Site	Distance Between Cation and NH_3 atoms			
		N	H	H	H
A	Na (1)	3.61	3.18	3.69	4.54
B	Na (2)	2.92	2.70	3.43	3.79
C	Na (1)	3.56	3.54	3.47	3.82
	Na (2)	2.77	3.10	3.25	3.49
D	Na (1)	3.36	2.97	3.72	4.29
	Na (3)	3.59	4.11	3.77	4.18
E	Na (2)	2.83	2.83	3.42	3.63
	Na (2)	3.05	3.63	3.63	3.00
F	Na (2)	3.24	2.85	3.17	4.19
	Na (3)	3.05	4.03	2.98	3.14

B. CO_2 Adsorption Isotherms. The potential parameters for CO_2 adsorption were fit to the experimental isotherm at 298 K and then used to calculate the isotherms at 277 K and 323 K. Figure 6 shows the adsorption isotherms for CO_2 on zeolite 4A at different temperatures and their comparison with the experimental isotherms from the manufacturer.[13] The agreement between the experimental and simulated isotherms is excellent for pressures greater than 3 kPa. At lower pressures there is a deviation of the simulated isotherms toward lower loadings. These results are also in agreement with the experimental isotherms obtained at different temperatures by Yucel and Ruthven.[27] Three different adsorption sites were identified for CO_2 in zeolite 4A. Binding site A is a single cation site in which the CO_2 molecule is coordinated to a Na(1) cation. In this site, the carbon dioxide molecule has its longitudinal axis aligned in such a way that one end points toward the cation and the other toward the center of the supercage. This is the most common site for adsorption at low pressures. Binding site B involves coordination of the adsorbed molecule with a Na(1) cation and a Na(2) cation. Each one of the oxygens from CO_2 is interacting with one of the coordinating cations. In this site, the longitudinal axis of the molecule is aligned in such a way that each end points to one of the coordinating cations, and the molecule is somewhat closer to the Na(1) than to the Na(2) cation. A carbon dioxide molecule in binding site C is coordinated to

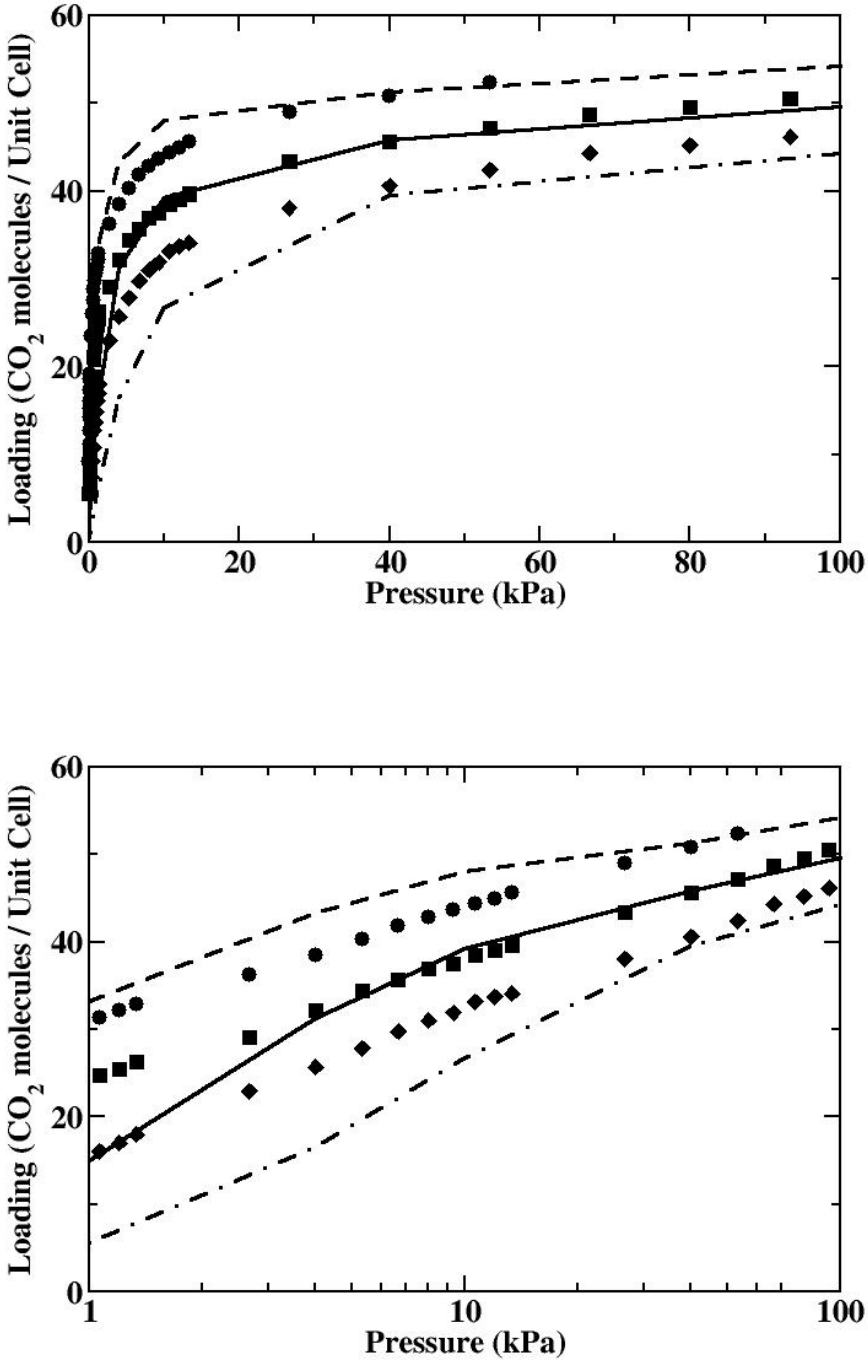

Figure 6. Experimental and simulated adsorption isotherms for CO_2 on zeolite 4A in linear (top) and logarithmic (bottom) scales. Experimental isotherms from the manufacturer[13] at 273 K (●), 298 K (■), and 323 K (▲). Simulated isotherms at 273 K (----), 298 K (——), and 323 K (·– –).

three cations each of a different type. One end of the longitudinal axis points to the Na(1) cation and the other toward a point in between the Na(2) and Na(3) cations. The distances from each end to the nearby coordinating cations are about the same. Sites B and C are more populated at the highest pressures studied here. Table 7 shows the average distances between cations and the atoms of carbon dioxide molecules adsorbed in zeolite 4A. The average oxygen-oxygen distance between neighboring CO_2 molecules in a saturated zeolite 4A is 3.75 Å.

Table 7. Average distances between cations and atoms of CO_2 adsorbed in zeolite 4A.

Binding Site	Cation Site	Distance Between Cation and CO_2 atoms		
		O	C	O
A	Na (1)	3.11	3.85	4.76
B	Na (1)	3.07	3.74	4.64
	Na (2)	5.15	4.14	3.21
C	Na (1)	3.27	3.83	4.62
	Na (2)	5.02	4.12	3.31
	Na (3)	4.78	4.40	3.08

C. H_2O Adsorption Isotherms. Two sets of potential parameters, SPC/E[14] and TIP3P,[15] were used to calculate H_2O adsorption. The potentials were used as given and no fitting was attempted as we are unaware of any experimental isotherms for comparison. Figures 7and 8 show the H_2O adsorption isotherms in zeolite 4A. The isotherms obtained using the two models are similar. Both sets of potentials provide saturation of H_2O at 298 K for pressures higher than 0.1 kPa with a saturation values of 235 and 243 H_2O molecules / unit cell of 4A for SPC/E and TIP3P potentials respectively. At 323 K saturation is reached at pressures higher than 0.5 kPa and at 373 K saturation is reached at about 100 kPa. The saturation value obtained is slightly higher than the 224 molecules H_2O / unit cell 4A given by Breck,[9] but within the10 % error bar of our simulation method. At pressures lower than the saturation pressure at each temperature, the simulated isotherms show a consistent higher adsorption of H_2O using the SPC/E potential than the TIP3P potential (Figure 9). At saturation, there are an average of 4.1 molecules of H_2O per -cage, which is in agreement with the simulation results of Faux *et al.*[2] (3.75 molecules per -cage) and the X-ray measurements of Gramlich and Meier.[18] Also at saturation, the average distance between an oxygen from H_2O and the closest Na cation is 2.75 Å and the average distance between two oxygen atoms from different water molecules is 2.75 Å. These results are in agreement with the findings of Faux *et al.*[2]

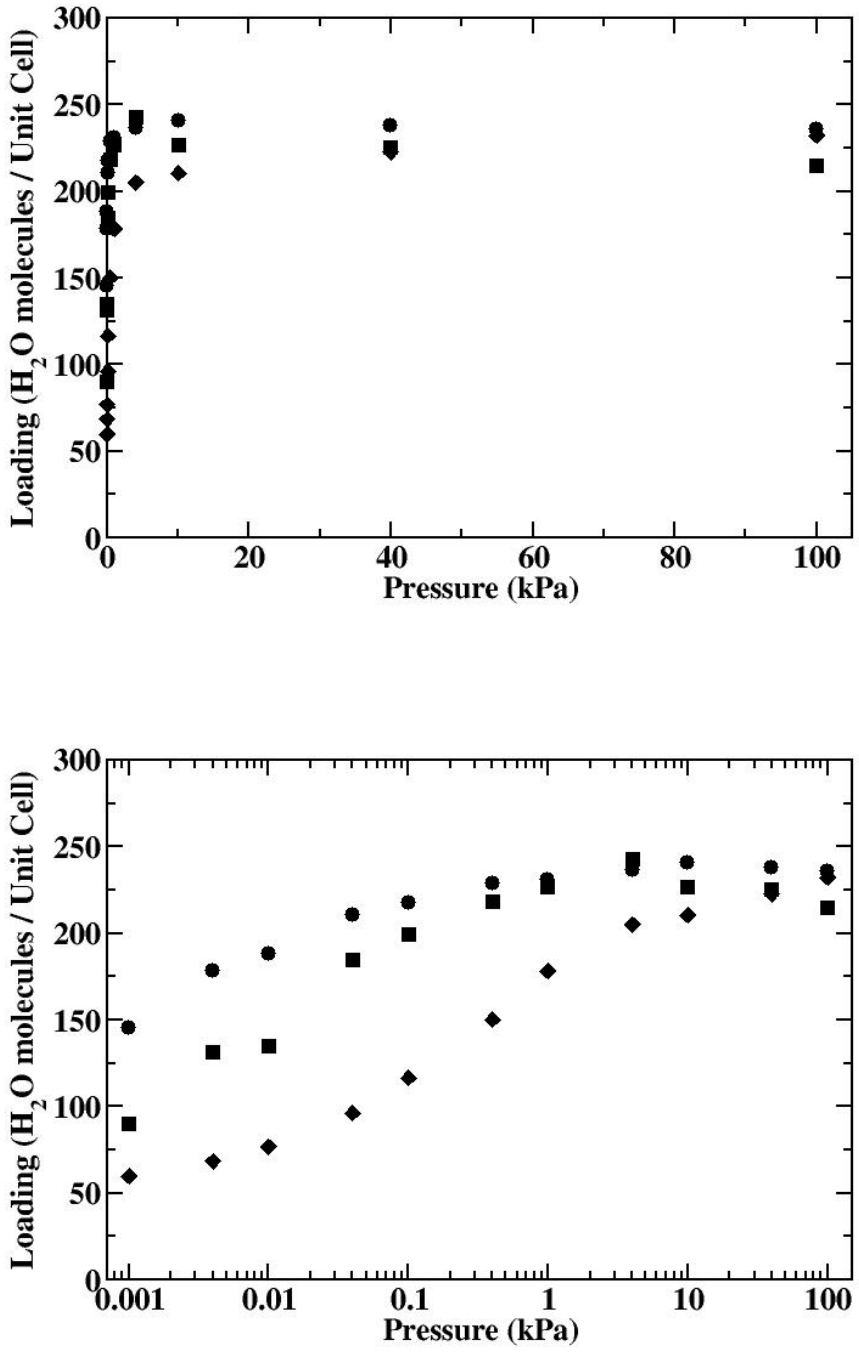

Figure 7. Simulated adsorption isotherms for H_2O on zeolite 4A using the SPC/E potential in linear (top) and logarithmic (bottom) scales at 298 K (circles), 323 K (squares), and 373 K (diamonds).

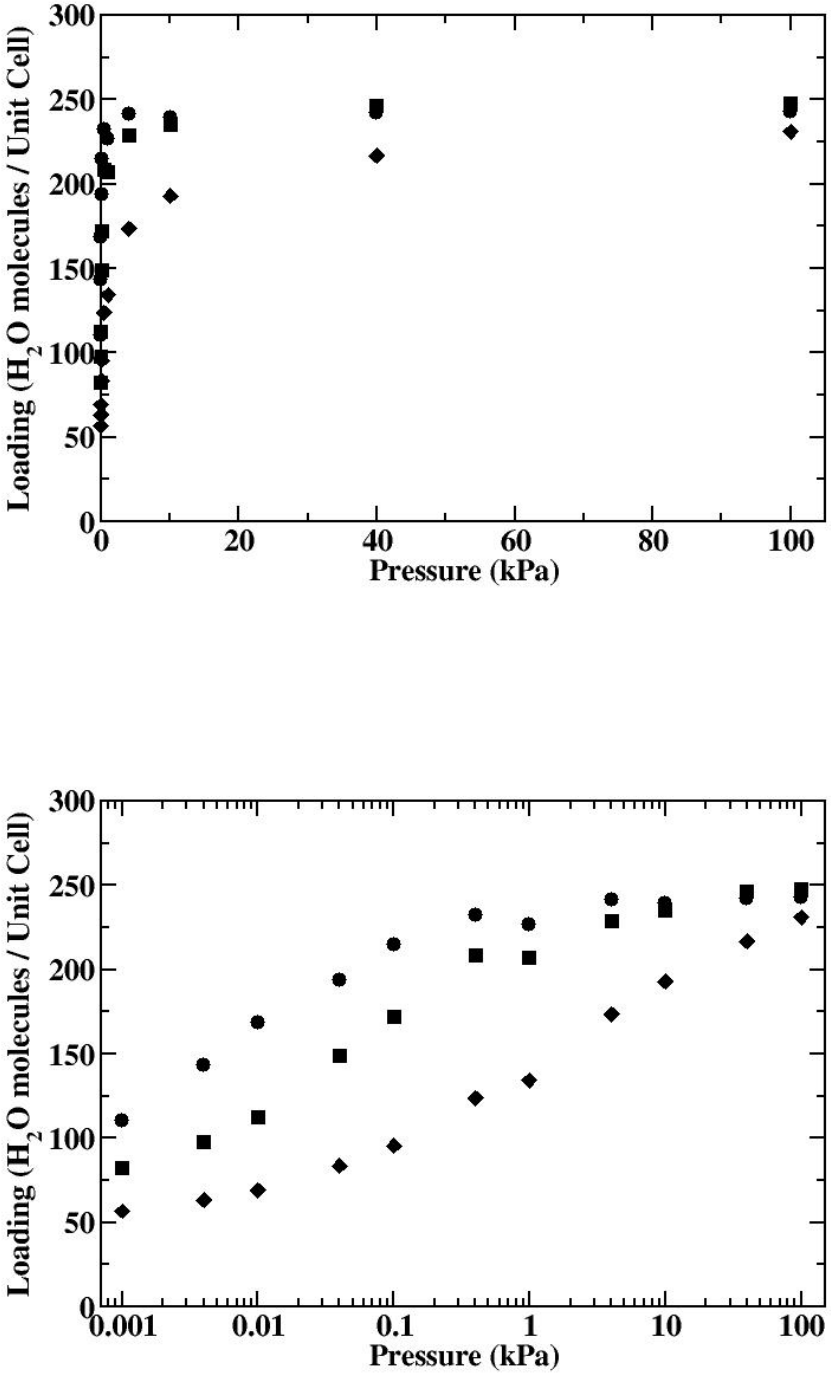

Figure 8. Simulated adsorption isotherms for H_2O on zeolite 4A using the TIP3P potential in linear (top) and logarithmic (bottom) scales at 298 K (circles), 323 K (squares), and 373 K (diamonds).

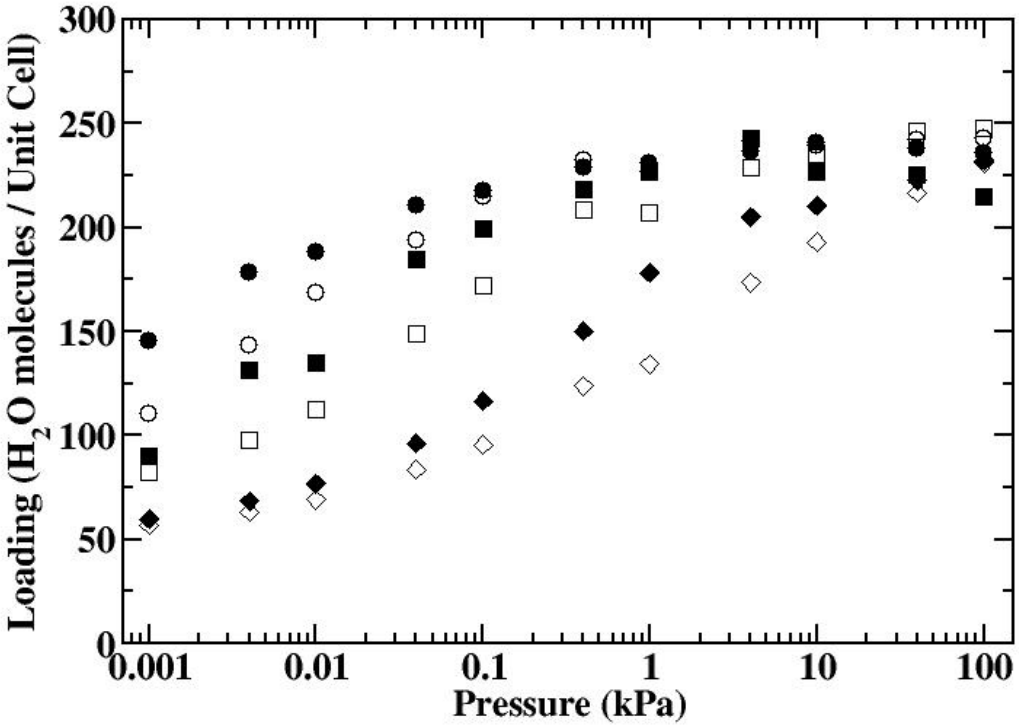

Figure 9. Comparison of the simulated adsorption isotherms for H_2O on zeolite 4A obtained using the SPC/E (filled symbols) and TIP3P (empty symbols) potentials at 298 K (circles), 323 K (squares), and 373 K (diamonds).

3.4 Summary of the Classical Force-Field Results

We have developed accurate potentials for the adsorption of NH_3 and CO_2 in zeolite 4A. We have obtained adsorption isotherms using these potentials along with existing potentials for water. There is a good agreement between the simulated isotherms and experimental ones. We have identified the preferred binding sites obtained using our classical potentials for NH_3, CO_2 and H_20 at high and low pressures.

4.0 Hybrid Quantum/Classical Method

A major goal of this project was the development of a multiscale approach bridging between quantum mechanical (QM) and classical methods. Although the bridging of length scales has been a very hot topic recently, there has been much less work on coupling QM and classical methods than on combining classical and continuum approaches. Attempts along these lines by the Kaxiras group at Harvard and the Vashishta group at Louisiana State University (LSU) have been based on terminating the boundaries of the QM region with hydrogen-like atoms. This approach has proven to be inefficient because, from the perspective of electronic structure, the hydrogen terminated boundary is not a good approximation to an extended system. As a result, a large QM buffer region around the region of interest is required, which is computationally very expensive. The LSU group reports that the time per atom for their QM calculations is 10^7 times greater than for their classical calculations. A typical classical simulation aimed at investigating collective phenomena in a material involves less than 10^7 atoms, so even for a single QM atom, the computational effort will be dominated by the QM part of the simulation. The LSU group also reports that they typically need a few hundred atoms in the QM region in order to isolate the phenomena of interest from the boundaries. Combined with the analysis above, this indicates that the time scales over which collective phenomena can be observed become extremely limited when a QM region is included using hydrogen-like termination.

A major step toward going beyond these limitations was recently taken by N. Bernstein at the Naval Research Laboratory. He noticed that, in a Green's-function-based linear-scaling electronic structure method, the Green's function elements at the boundary of a QM system could easily be constrained to values that approximate the embedding of the system within an extended QM system. Since QM calculations for the embedding system are unnecessary, it can be modeled using classical techniques. In essence, Bernstein's approach provides a method of making a smooth crossover between QM and classical regions. Since Bernstein's approach provides a much better approximation to an extended system at the boundaries, it allows the size of the buffer zone to be reduced without sacrificing accuracy.

Bernstein's approach is based on a simplified, empirical QM model (tight-binding), which limits its applicability to a few, relatively simple systems where high quality tight-bonding models are available. In general, such models are not available for the systems that interest Sandia. Therefore, we started with Bernstein's basic idea and developed a new approach that allows the QM region to be represented using first-principles methods based on the Kohn-Sham density functional theory (DFT). Since the DFT has been shown to give reliable results for a huge variety of systems, the techniques developed in the first part of this project should be applicable to bridge between the classical and QM length scales in a wide variety of systems.

The generalization of Bernstein's idea to first-principles methods presented a number of challenges. First, modern DFT calculations predominantly use a flexible, grid-based representation of the electronic structure. A straightforward Green's-function-based approach using such a representation is not practical because they require a large number of basis functions per atom, and therefore the Green's function (which is a matrix in the basis representation) becomes unmanageably large. Goedecker has shown how to overcome this limitation by using a Fermi operator projection method in which the calculation of the entire Green's function is replaced with

the calculation of matrix elements of the Green's function involving a small number of trial vectors. We adapted this version of the Green's-function-based electronic structure method for use in our multiscale calculations, applied boundary conditions that represented the embedding of the QM system within the classical system, integrated the two programs that perform quantum mechanical and classical calculations, investigated several optimizations, and tested the resulting approach.

4.1 Green's-Function-Based Electronic Structure

Our initial work developed a QM model based on the DFT that provides a localized representation of electronic structure. This method replaces the extended electronic eigenstates used in conventional DFT calculations with a set of localized Wannier-function-like electronic states that span the same subspace. We implemented a Green's-function-based electronic structure (GFES) method since a smooth crossover between QM and classical regions can be achieved by imposing boundary conditions on the Green's-function that approximate the embedding of the system within an extended QM system. Due to the large number of basis functions per atom associated with the flexible, grid-based approach predominantly used by modern DFT codes, a straightforward Green's-function-based calculation of the QM density matrix ρ is not practical. Instead, we have implemented a Fermi operator projection method. This approach constructs a set of orbitals by applying the density operator (represented by the integral of the Green's function around a contour in the complex plane) to a set of linearly independent trial vectors. The density operator projects into the occupied subspace of the DFT Hamiltonian, and since the resulting orbitals span the occupied subspace, the density matrix ρ can be efficiently and straightforwardly represented in terms of the orbitals. For an operator A, the expectation is given by $\langle A \rangle = Tr(\rho A) = Tr(D\hat{A})$, where $\hat{A}$ contains the matrix elements of A with respect to the orbitals, and D is the inverse of the orbital overlap matrix. Since the density operator is local in space, we obtain localized orbitals from localized trial functions. Figure 10 shows the four orbitals associated with an oxygen atom in our model structure. Gaussian trial functions with radius 1.2 A.U. were used to construct the orbitals. The centers of trial functions were taken to be the points of a distorted tetrahedron centered on the oxygen and with two of its corners at the Si-O and Al-O bond centers. As expected for a wide band gap system such as Zeolite-4A, the orbitals decay rapidly in space and provide a strongly localized representation of the electronic structure.

The density operator is represented by the integral of the Green's function around a contour in the complex energy plane that includes only the occupied Kohn-Sham eigenvalues. In practice, this integral is evaluated numerically.The convergence of the band energy with respect to the number of quadrature points is shown in Fig. 11. Notice that the band energy is variational, and that the convergence is exponential. In practice, results converged to 1 meV are typical, which would require about 14 points using the current scheme.

4.2 Boundary Conditions for Embedding

Our next task was to apply boundary conditions on the QM method to approximate the embedding of the QM calculations within a larger system. First, we constrained the GFES algorithm to an approximate result in the quantum/classical interface region. This is accomplished by freezing the localized electronic states in the interface region at the values obtained from a fully QM calculation for a larger reference system that includes both the quantum and interface regions. Since many

hybrid quantum/classical calculations are performed for each reference calculation, the computational expense of these fully QM calculations is not overly burdensome. To our knowledge, reference calculations have not previously been used in this manner, and we believe that this novel approach will greatly extend the flexibility of our quantum/classical method

In order to incorporate electrostatic interactions between the classical and quantum regions, we created special atomic data files for the Socorro electronic structure software that represent the classical atoms. These new "pseudoatoms" have local charges determined from the net charges of the classical atoms. As a result, there are no electrons associated with the bonds between these atoms. When these special atoms are included in the crystal structure, Socorro automatically calculates all electrostatic interactions relevant to the combined quantum and classical systems.

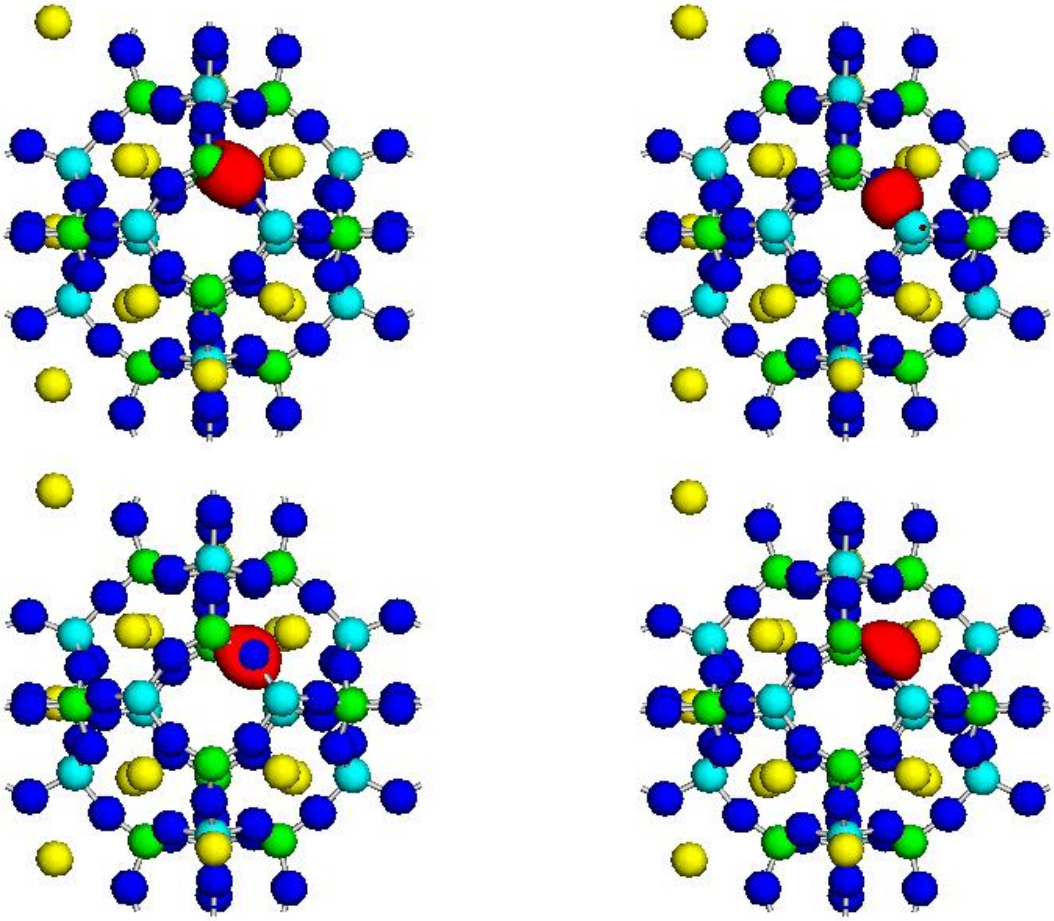

Figure 10. Localized orbitals obtained with the GFES method. The red isosurface indicates 1% of the maximum value for each orbital. The color scheme is the same as Fig. 2, but a different region of the structure is shown. The orbitals correspond to (a) a Si-O covalent bond, (b) a Al-O covalent bond, (c) and (d) oxygen lone pairs directed away from the observer, and (d) an oxygen lone pair direct toward the upper right.

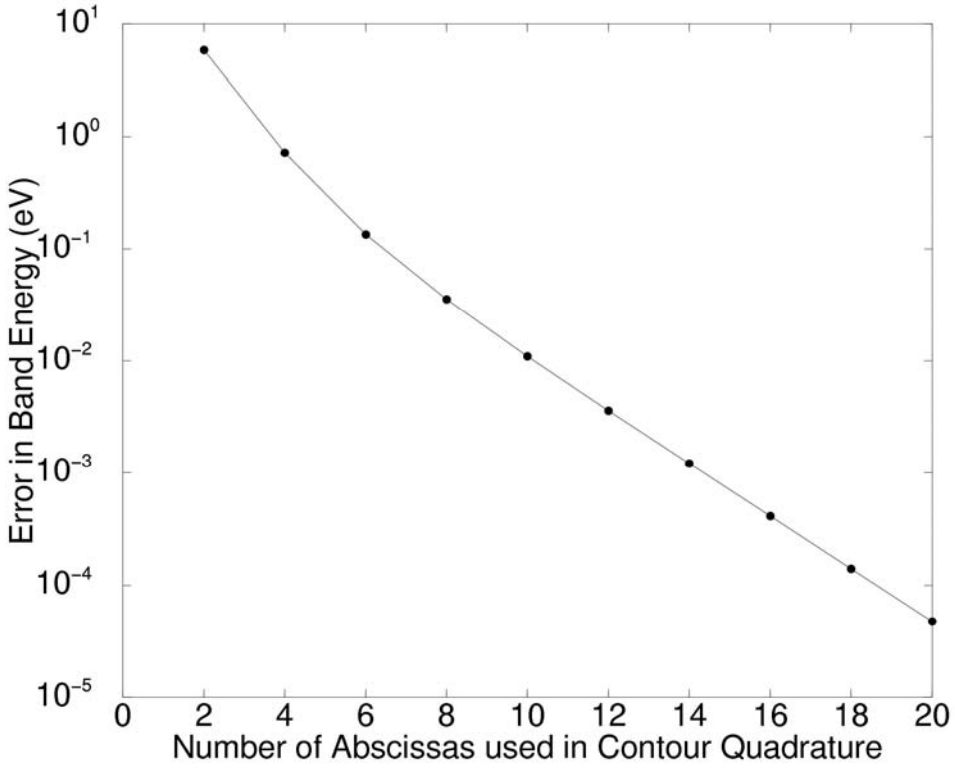

Figure 11. Convergence of the GFES electronic energy of our model structure with respect to the number of quadrature points used to perform the contour integral. The error is the small difference between the GFES result and the exact result (-2865.802 eV), not the fractional error (even smaller). Evenly spaced abscissas on a circular contour were used..

4.3 Linking of the Classical and Quantum Codes

The final step to embedding our quantum mechanical calculations inside a dynamic background represented using classical force-fields was to calculate the local interactions associated with atoms at the quantum/classical boundary and in the classical region. The calculation of these forces was achieved by linking of the classical simulation code LAMMPS with the quantum mechanical code Socorro. Initial work involved restructuring LAMMPS by creating an object within LAMMPS that isolates the data that must be shared between LAMMPS and Socorro. Another important activity was merging the input files used by the two codes. Approaches were developed for passing control and apportioning the calculations between the two codes. Finally, we developed code to allow coexistence of the two different parallelization strategies used by LAAMPS and Socorro.

Data Compartmentalization in LAMMPS

The version of LAMMPS used in the hybrid code is LAMMPS2001, which is predominantly a Fortran77 code. This code uses a small amount of Fortran90 to allow for dynamic memory allocation, but there is no data compartmentalization in LAMMPS2001 as supplied. In contrast, Socorro is written in order to fully utilize data structures, objects and modules, and all data is

tightly compartmentalized. The decision was therefore made that before linking the codes, it would be necessary to convert LAMMPS to have data compartmentalization, particularly with regards to data that would be shared between the two codes. The hybrid code uses a large object called *structure_snapshot* which encompasses all the information necessary to have a snapshot of the system at a point in time. The object is reproduced in Appendix A, where it can be seen that all data are now dereferenced and explicit calls to members are not allowed. Rather it is necessary to use accessor functions such as *get_var(strucuture)* and *set_var(structure, value)* in order to retrieve or set values. The reasoning behind this is twofold. First, this design allows LAMMPS and Socorro to keep the bulk of their data separate, avoiding concerns regarding namespace conflicts. The only data that are shared are those that are included in the shared object, *structure_snapshot*. The second reason for this design is to ensure expandability. It is not only possible to add additional data structures to *structure_snapshot* to allow for more advanced features, such as bond breaking and formation, but it is also possible to utilize different molecular dynamics packages with more advanced functionality (such as GRASP), since the nature of the object does not specifically depend on the structure of LAMMPS.

Merged Input Files

The data that are included in the object reflect the information that needs to be transferred between the two codes. In general, LAMMPS require two types of information for a simulation; (1) the specifics of the simulation itself, including time step, thermostat, force-fields, processor grid, etc. and (2) the specifics of the system, including atom types, positions, velocities, forces, charges, etc. The first type is not necessary for the running of Socorro, because it applied to the molecular dynamics simulation only. Socorro also requires two types of information for a calculation; (1) the specifics of the DFT calculation, such as number of bands, energy cutoff, etc. and (2) specifics of the system, such as atom types, locations, etc. It is clear that the second type of information for both codes is the information that must be shared and transferred between the two codes. It is necessary to first understand the basics of the input files for the two codes before it is possible to understand the merging strategy.

LAMMPS uses two different files, the *input* file and the *data* file. The *input* file is essentially the master file for the simulation, and contains the instructions regarding the simulation itself, such as time step, number of steps, temperature, etc. Included in the *input* file is a command that instructs LAMMPS to read the *data* file. The *data* file contains the information about the system itself, including atom types, positions, bonds, angles, dihedrals, etc. This separation of files allows the user to keep the simulation details unchanged while modifying the system itself, whether this is for simulating multiple different systems, or simply using a restart file.

The input files for Socorro are similar to those for LAMMPS in that there is a input file, *argvf*, which contains information regarding the parameters of the calculation, and a data file, *crystal*, which contains the atom types, positions, and lattice parameters. Socorro also reads in pseudopotential files for each atom type represented in the *crystal* file.

It was determined that while the bulk of the data required by Socorro are also required by LAMMPS, LAMMPS requires more additional information than Socorro. Therefore it was decided that the Socorro files should be merged into the LAMMPS files as much as possible. To this end

the hybrid code uses the Socorro *argvf* file to control the DFT calculation, the LAMMPS *input* file to control the MD simulation, and the LAMMPS *data* file to give all information about the system itself, except for pseudopotentials. The LAMMPS *data* file format was modified to contain additional information. First, each atom is identified as either quantum or classical; this information is passed to the MD integrator for decisions regarding what interactions are applied and which are discarded. Second, each atomic type is associated with a pseudopotential file; purely classical atoms have a simple pseudopotential that allows for representation as a smeared charge cloud, while quantum atoms have a standard pseudopotential. Third, lattice information for Socorro is included in addition to a classical box size; in most instances this information is redundant, but it has been kept separate for special cases (e.g. when a hexagonal unit cell is used). The reading in and transferring of data will be covered in the next section.

Passing of Control

After the merging of the input files, the next major step in the linking of the codes is the passing of control from one code to the other. With the decision made to merge the Socorro system information into the LAMMPS files, it followed logically that initial control of the simulation should belong to LAMMPS, which then passes control to Socorro. The general procedure is as follows:

1. LAMMPS is initialized, and reads in the *input* and *data* files.
2. LAMMPS performs its initial setup of the MD simulation, and starts the integration loop.
3. After step 0 of the integration, necessary for initial energy and force calculations and to seed the velocity Verlet integration scheme, LAMMPS calls the initial crystal constructor for Socorro, and relinquishes control.
4. A rewritten Socorro crystal constructor populates the Socorro variables from the *structure_snapshot* object, rather than from the *crystal* data file.
5. Socorro reads in the *argvf* and pseudopotential files for the specifics of the DFT calculation.
6. Socorro performs a minimization calculation, proceeding until the energy has converged.
7. LAMMPS regains control and adds the forces calculated by Socorro to the forces calculated from the force field.
8. LAMMPS moves atoms according to the total forces, updating atom arrays.
9. LAMMPS updates the integration step, and passes new atomic position information to Socorro.
10. Go to step 6.

Division of Labor

The work involved in calculating the forces is shared by the two codes. All atoms in a given simulation are designated as either classical or quantum atoms by an identifier in the data file. Classical interactions (i.e. Lennard-Jones type nonbonded interactions as well as bonds, angles, and dihedrals) are calculated between all atoms by LAMMPS. It is therefore necessary for all atoms, including quantum atoms, to be described by a classical force field. Calculated forces are included in the calculation if at least one atom is classical, and discarded otherwise. Socorro is responsible for all electrostatic interactions, as well as all interactions between purely quantum atoms.

Integrated Parallelization

Both LAMMPS and Socorro are massively parallel codes that can spread calculations over a large number of processors. The difference, however, lies in how the work is divided amongst those processors. LAMMPS is spatially decomposed (i.e. each processor only stores information about atoms in a given volume of simulation space), which leads to a restriction on the maximum number of processors that can be used for a simulation. The restriction is due to an interplay between the cutoff imposed for interatomic interactions and the size of the domain assigned to a processor; essentially processors should not be assigned domains that are smaller than twice the cutoff to avoid issues with the construction of neighbor lists. Socorro decomposes domains in Fourier space, so that each processor must have information about all atoms, but only certain volumes of k-space. This results both in a problem and a benefit. The problem arises because LAMMPS must communicate all atomic information to all processors before passing control to Socorro, and then redistribute appropriate force information to each processor after Socorro's minimization step. This results in extra memory requirements for the classical part of the calculation. The benefit of this difference in decomposition is that LAMMPS and Socorro can be allowed to run on different numbers of processors. This is a benefit because a problem that is large for Socorro (e.g. hundreds of atoms), requiring many processors, can actually be small for LAMMPS and possibly only be run on one processor. This required modification of LAMMPS, as the standard version performs checks to ensure that the number of processors allocated by the MPI calls matches the processor grid assigned in the input file. LAMMPS is now configured to divide atoms amongst the assigned processor grid, while allowing Socorro to divide up work as necessary on all allocated processors.

4.4 Optimization of Electronic Structure Algorithms

Since a key goal of our hybrid quantum/classical approach is to reduce the computational effort required to treat large systems, optimization of the approach was an important consideration. The computational requirements of the method are dominated by obtaining the localized electronic states. Therefore, we implemented and explored alternative algorithms for accomplishing this task. The first alternative algorithm is based on the Grassmann Conjugate Gradient (GCG) algorithm, and its performance is compared to the GFES method in Fig. 12.

For calculations in which we do not force each electronic state to be strictly zero outside some region (strict localization), we find that GCG is much more efficient. However, when we impose strict localization, we find that GCG stops converging. This behavior can be understood in terms of the effects of localization on the condition number of the minimization problem involved in the GCG method. In contrast, GFES converges at least as well in the strictly localized case as in the nonlocalized case. Another important convergence criteria for such methods is the convergence of the total energy with respect to the radius of the strict localization spheres (shown in Fig. 13). In this regard, we find that GCG converges substantially faster than GFES. This results from the fact that GCG finds orbitals that minimize the energy within the given constraint. These results suggest that further work (possibly involving a combination of the two methods) should enable a substantial improvement in the efficiency of the QM method.

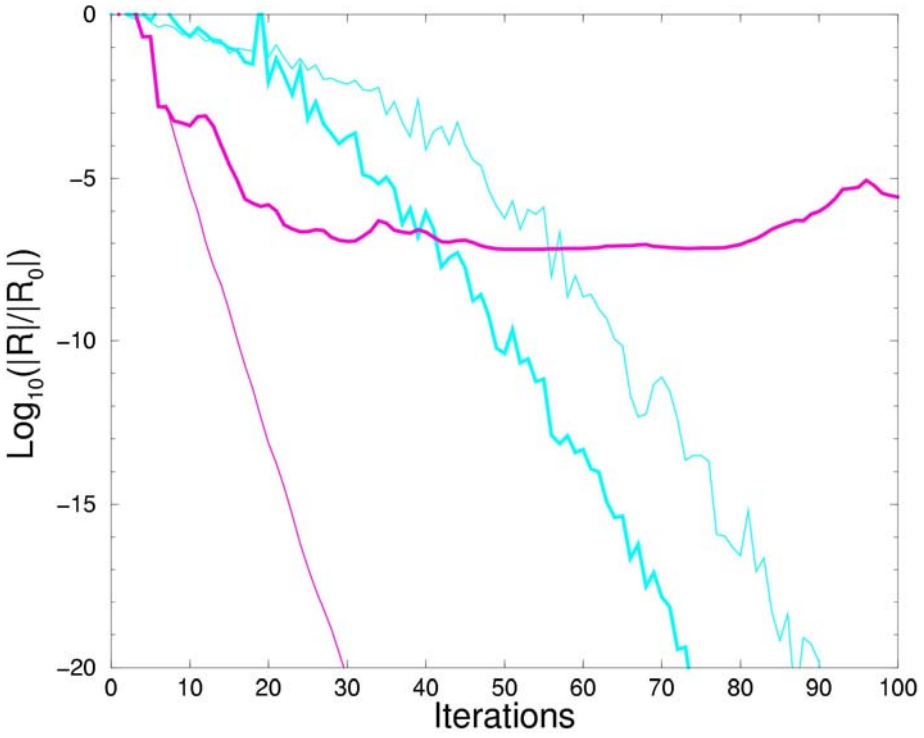

Figure 12. Convergence of the GFES (cyan) and Grassmann Conjugate Gradient (magenta) methods with respect to the number of iterations. The thin lines are without strict localization, and the thick lines are with strict localization at 4.3 Å.

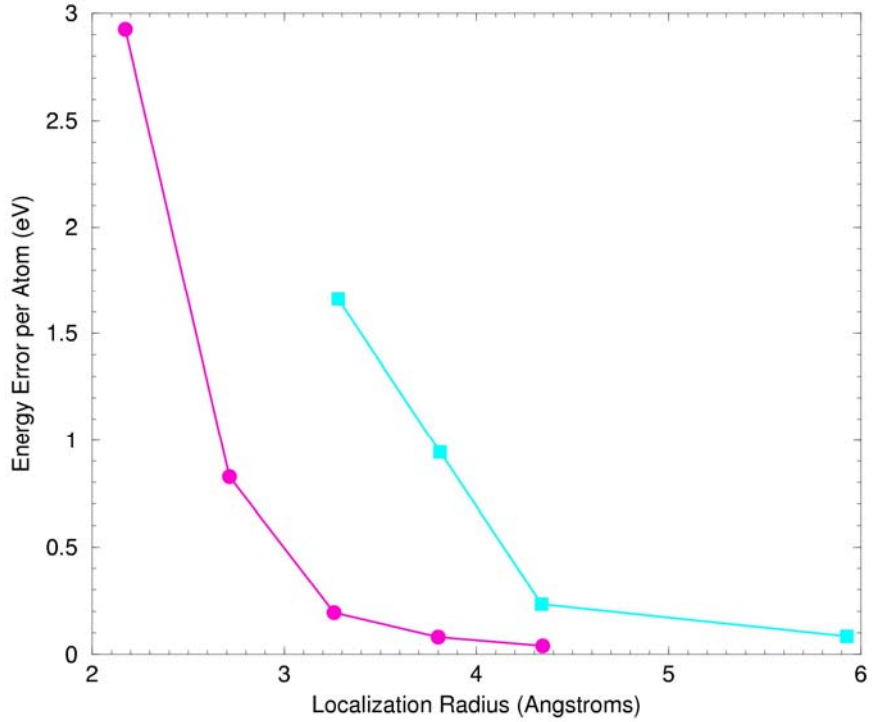

Figure 13. Convergence of the energy with respect to the size of the localization region using the GFES (cyan) and Grassmann Conjugate Gradient (magenta) methods.

Instead of focusing additional effort on further optimization of general purpose algorithms such as GFES and GCG, we pursued another approach that takes advantage of the specific physics of our system of interest. In this approach, the localized orbitals in the quantum/classical boundary region that provide boundary conditions for the electrons in the QM region are replaced with simple electrostatic charges. In oxide materials (such as zeolites), the large negative charge on the oxygen atoms prevents significant penetration by additional electrons into their vicinity. Therefore, we found that this relatively crude treatment of the boundary conditions was adequate to achieve good results. Furthermore, this approach naturally localizes the electronic wavefunctions to the QM region, and standard electronic structure algorithms can be used. We found the resulting scheme to be both efficient on Sandia's parallel computers and significantly easier to use. Another important part of refining the algorithms and their implementation was the systematic reduction of memory usage. This proved to be critical to the simulation of large systems such as the full Zeolite-4A unit cell. Two particularly severe memory hot-spots were identified and eliminated.

4.5 Testing of the Hybrid Quantum/Classical Approach

Figure 14 shows the structures obtained from our quantum/classical approach for absorption of a water molecule into one and two ion windows. Comparison to Fig. 4 shows that there are almost no visually apparent differences between the results of the two methods. The preferred absorption sites and energy differences obtained from our various models are summarized in Table 8. As mentioned earlier, neither the purely classical treatment nor the one-eighth-cell quantum model are adequate to accurately capture the behavior of this system. The purely classical approach fails to capture the details of the local chemical behavior close to the water molecule, while the 1/8 cell model does not capture the collective relaxation of the zeolite. In contrast, the hybrid model and the one-half-cell quantum model give fairly good agreement with the full quantum results. This is because both of these models adequately capture both the local chemical behavior and the long range collective behavior characteristic of this system. The computational resources required to perform the calculations using each model are also shown in Table 8. The quantum/classical approach is a factor of 11 more efficient than quantum calculations for the whole system and a factor of 4 more efficient than calculations for the one-half-cell model.

Table 8. Comparison of the results and computational cost of various methods and models.

Method	Site	Energy (meV)	Time (node-hr)
Classical 8	1-ion	461	46
QM 1/8	1-ion	15	2878
QM 1/2	2-ion	511	40,311
QM Full	**2-ion**	**450**	**111,876**
Hybrid	2-ion	392	10,285

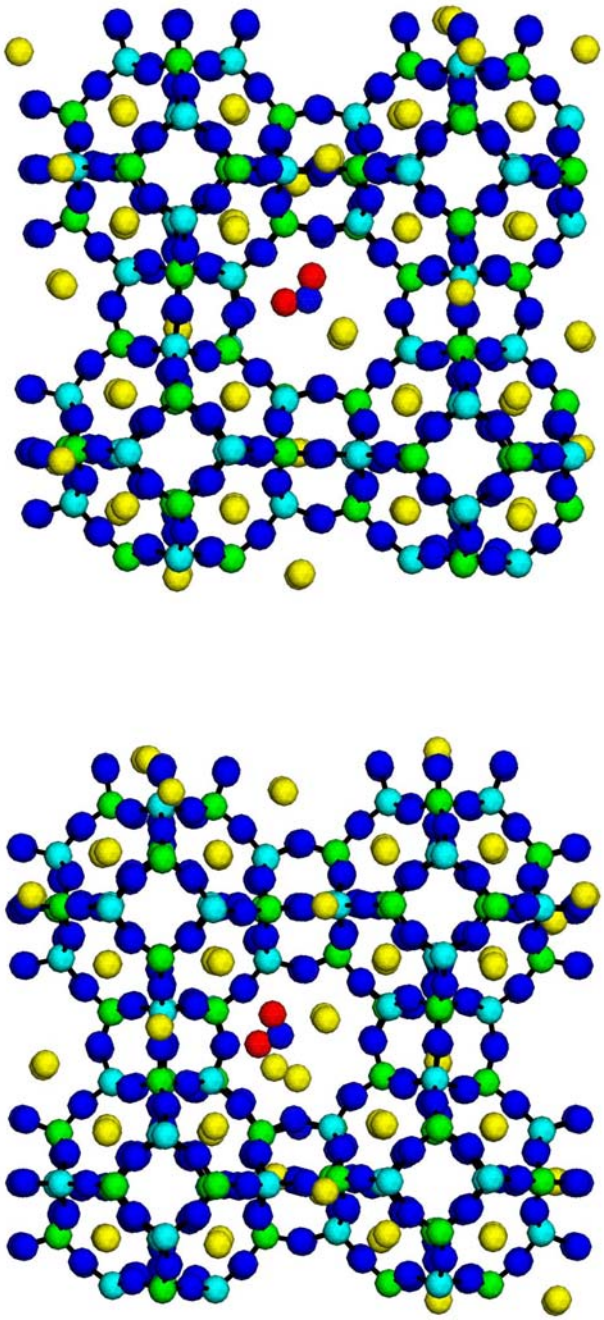

Figure 14. Results of multiscale calculations for a water molecule absorbed into Zeolite-4A in (a) a 1 ion window and (b) into a 2 ion window. The dark blue balls indicate O atoms, the light blue balls indicate Al atoms, the green balls indicate Si atoms, the yellow balls indicate Na atoms, and the red balls indicate hydrogen atoms.

5.0 Proposed Mechanisms of Desiccant Aging

Zeolites from the LTA family (3A, 4A, and 5A, where the *n* in *n*A refers to the pore size in Å) are commonly used as industrial desiccants. While the ubiquitous silica gel desiccants perform adequately for general use, their amorphous structure leads to the adsorption of a large number of molecules in addition to water. The LTA zeolites, on the other hand, exhibit strong preferential adsorption for water because of their small pore openings, as well as the polar nature of the zeolite cage itself. This affinity for water makes these zeolites ideal desiccants for applications in which moisture control is crucial, such as the packaging of sealed electronics. The presence of water in electronic assemblies can lead to corrosion, conduction through dendritic growth, and even conduction through the water itself, if large quantities are present. In addition to water, however, the LTA zeolites adsorb a number of small molecules, including O_2, N_2, NH_3, CO_2 and others. It is therefore a concern that reactions among adsorbed molecules, such as $NH_3 + CO_2 + H_2O \rightarrow NH_4^+ + HCO_3$ may occur within the zeolite, leading to a reduction in water capacity.

In the course of this project, we identified and investigated two possible mechanisms for the reduced capacity of Zeolite-based desiccants after aging in the presence of water and ammonia. In the first proposed mechanism, ammonium ions substitute for the sodium ions located in the 8-oxygen rings of the Zeolite structure. The large ammonium ions then block the diffusion of water through these rings. Calculations showed that the presence of an ammonium ion does indeed increase the energy of a water molecule as it moves through the window. In the second proposed mechanism, hydroxide ions, created by the reaction of water and ammonia, catalytically hydrolyze bonds in the zeolite framework leading to degradation of the zeolite. A mechanism for this process was explored.

5.1 NH_4^+ Mechanism

The combination of absorption of small molecules within the 8-oxygen rings and crowding effects within these rings suggested our first possible explanation for the irreversible aging observed in zeolite-based weapons desiccants. It is believed that these desiccants have been exposed to ammonia and carbon dioxide in addition to water. These species are likely to react to form ammonium and carbonate ions. If the resulting ammonium ions absorb into the 8-oxygen rings, they are likely to replace Na ions and become immobilized in the Zeolite structure. Furthermore, the large size of the ammonium ion (relative to Na) makes it likely that such an absorbed ion will block the diffusion of water through the window formed by the 8-oxygen ring, and hence through the Zeolite structure. We investigated this hypothesis by studying the absorption of ammonia and ammonium in Zeolite-4A. The results are shown in Fig. 15. The absorption structure for the ammonia molecule (a) is very similar to our results for water. The ammonia molecule makes two hydrogen bonds to oxygens from the 8-oxygen ring. The ammonium ion (b) is obtained by adding another hydrogen atom and removing a Na ion in another window to allow the ion to have its usual +1 charge. In (c), ammonium replaces Na. We proceeded to study the diffusion of water in the presence of an absorbed ammonium ion (d) and found that the presence of the ammonium ion raises the energy of a co-absorbed water by 0.3 eV. It is likely that ammonium substitution into an 8-oxygen ring that previously contained two Na ions may block diffusion of water even more strongly due to collective crowding by the ammonium ion and the remaining Na ion.

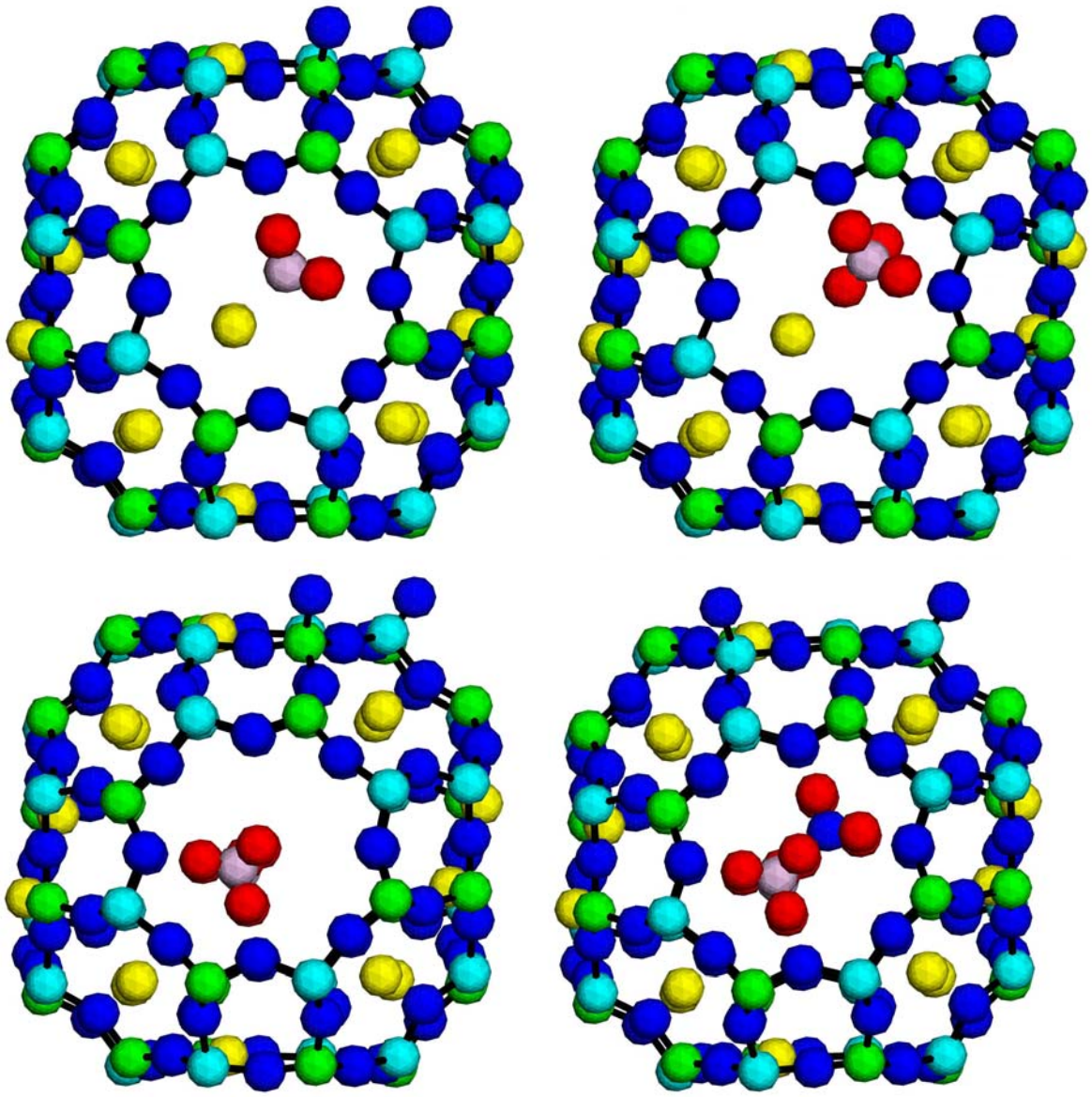

Figure 15. Low energy structures for (a) an ammonia molecule and (b) an ammonium ion absorbed in our model structure. In (c), the ammonium ion replaces the Na ion, and in (d), the ammonium ion raises the energy of a co-absorbed water molecule. The dark blue balls indicate O atoms, the light blue balls indicate Al atoms, the green balls indicate Si atoms, the yellow balls indicate Na atoms, the red balls indicate H atoms, and the purple ball indicates a N atom. Both molecules are located in an 8-oxygen ring containing a single Na ion. In (a), a third hydrogen atom is located behind the nitrogen atom where it can not be seen.

5.2 OH^- Mechanism

When ammonia reacts with water to form ammonium, it also produces hydroxide (OH^-) ions. This observation led us to our second possible mechanism for the aging of Zeolites. We investigated a series of reactions by which the hydroxide ion could act as a catalyst for hydrolytic degradation of the zeolite cage. The intermediate structures occurring during this process are shown in Fig. 16

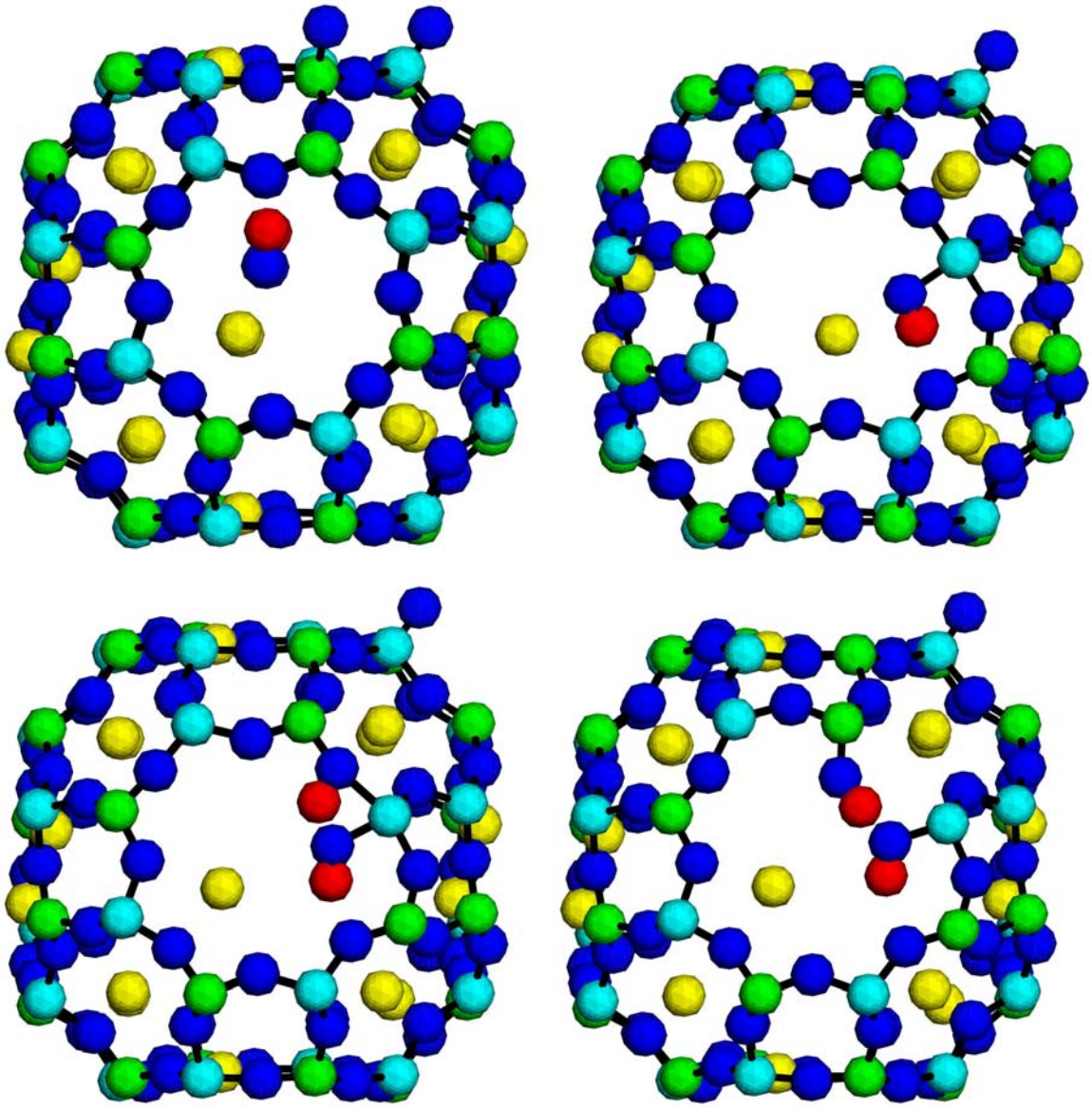

Figure 16. Low energy structures for (a) an OH^- ion in unbound configuration, (b) an OH^- ion in a bound configuration, (c) the activated oxygen captures a hydrogen ion, and (d) bonds in the zeolite cage are broken. The dark blue balls indicate O atoms, the light blue balls indicate Al atoms, the green balls indicate Si atoms, the yellow balls indicate Na atoms, and the red balls indicate H atoms.

We found that the hydroxide ion normally sits with its hydrogen atom bonded to an oxygen atom from the 8-oxygen ring, and its oxygen atom pointing toward the Na ion (a). However, we also found an alternative configuration (b) in which the oxygen atom is bonded to an aluminum atom from the 8-oxygen ring. This creates a five coordinated aluminum atom, which logically weakens the bonds between the aluminum atom and its neighboring oxygen atoms. If the excess electron is transferred from the hydroxide ion to one of the neighboring oxygens, the bond between the aluminum atom and that oxygen should easily break. At this point, it should be easy for the neighboring oxygen atom to interact with a nearby water molecule (not shown) exchanging its

excess electron for a hydrogen atom and recreating the hydroxide ion (also not shown). The resulting configuration (c) has two OH groups: one bonded to the aluminum atom, and one bonded to the next silicon atom around the 8-oxygen ring. As rearrangement of this structure (d) moves the second OH group away from the Al atom and allows hydrogen bonding between the OH groups. Repeated application of this process (activated by the regenerated hydroxide ion) could eventually lead to severe degradation of the zeolite cage.

6.0 Conclusions and Future Work

We developed a hybrid quantum/classical method able to accurately portray chemical reactions as well as collective phenomena and applied it in concert with purely quantum and purely classical techniques to identify two possible mechanisms for the aging of Zeolite-based desiccants. One objective of this project was to help establish Sandia in the exciting field of multiscale modeling, and our success is this arena is demonstrated by an invitation to present the results of this project at the March Meeting of the American Physical Society. Our proposed mechanisms for zeolite aging show the ability of this work to contribute to Sandia's Stockpile Stewardship mission, and continuing work applying our approach to investigate the aging of desiccants is funded under the ASCI program. While we have chosen one specific application here, over time, the capabilities that we have developed should be useful across a wide range of important Sandia programs. One initial example that is currently in progress is the application of these techniques to study the binding of self-assembled monolayers to silicon dioxide surfaces.

7.0 References

(1) Lee, S. H.; Moon, G. K.; Choi, S. G.; Kim, H. S. *J. Phys. Chem.* **1994**, *98*, 1561.

(2) Faux, D. A.; Smith, W.; Forester, T. R. *J. Phys. Chem. B* **1997**, *101*, 1762.

(3) Faux, D. A. *J. Phys. Chem. B* **1998**, *102*, 10658.

(4) Faux, D. A. *J. Phys. Chem. B* **1999**, *103*, 7803.

(5) Koh, K. O.; Jhon, M. S. *Zeolites* **1985**, *5*, 313.

(6) Mizukami, K.; Takaba, H.; Kobayashi, Y.; Oumi, Y.; Belosludov, R. V.; H.; Takami, S.; Kubo, M.; Miyamoto, A. *J. Membr. Sci.* **2001**, *188*, 21.

(7) Löwenstein, M. *Am. Mineral.* **1954**, *39*,92.

(8) Pluth, J. J.; Smith, J. V. *J. Am. Chem. Soc.* **1980**, *102*, 4704.

(9) Breck, D. W. In *Zeolite Molecular Sieves*; Wiley: New York, 1974.

(10) Helminen, J.; Helenius, J.; Paatero, E.; Turunen, I. *AIChE J.* **2000**, *46*, 1541.

(11) Helminen, J.; Helenius, J.; Paatero, E.; Turunen, I. *J. Chem. Eng. Data* **2001**, *46*, 391.

(12) Martin, M. G.; Siepmann, J. I. *J. Phys. Chem. B* **1999**, *103*, 4508.

(13) Davison 4A Molecular Sieves Technical Product Data. Source: Adsorbents Department, Davison Chemical Division, W. R. Grace & Co., Baltimore, MD.

(14) Berendsen, H. J. C.; Grigera, J. R.; Straatsma, T. P. *J. Phys. Chem.* **1987**, *91*, 6269.

(15) Jorgensen, W. L.; Chandrasekhar, J.; Madura, J. D.; Impey, R.; W.; Klein, M. L. *J. Chem. Phys.* **1983**, *79*, 926.

(16) Rizzo, R. C.; Jorgensen, W. L. *J. Am. Chem. Soc.* **1999**, *121*, 4827.

(17) Makrodimitris, K.; Papadopoulos, G. K.; Theodorou, D. N. *J. Phys. Chem. B* **2001**, *105*, 777.

(18) Gramlich, V.; Meier, W. M. Z. *Kristallogr.* **1971**, *133*, 134.

(19) Brändle, M.; Sauer, J. *J. Mol. Catal. A* **1997**, *119*, 19.
(20) Kiselev, A. V.; Du, P. Q. *J. Chem. Soc., Faraday Trans. 2* **1981**, *77*, 1.
(21) Demontis, P.; Kärger, J.; Suffritti, G. B.; Tilocca, A. *Phys. Chem. Chem. Phys.* **2000**, *2*, 1455.
(22) Goj, A.; Sholl, D. S.; Akten, E. D.; Kohen, D. *J. Phys. Chem. B* **2002**, *106*, 8367.
(23) Hirotani, A.; Mizukami, K.; Miura, R.; Takaba, H.; Miya, T.; Fahmi, A.; Stirling, A.; Kubo, M.; Miyamoto, A. *Appl. Surf. Sci.* **1997**, *120*, 81.
(24) Karavias, F.; Myers, A. L. *Mol. Simul.* **1991**, *8*, 23.
(25) Karavias, F.; Myers, A. L. *Mol. Simul.* **1991**, *8*, 51.
(26) Potoff, J. J.; Siepmann, J. I. *AIChE J.* **2001**, *47*, 1676.
(27) Yucel, H.; Ruthven, D. M. *J. Colloid Interface Sci.* **1980**, *74*, 186.

8.0 Appendix A: Code for LAMMPS/Socorro Interface

```
!This is the main object containing all the structural information
!for both LAMMPS and SOCORRO

     MODULE structure_snapshot

        IMPLICIT NONE
        PRIVATE

     !lifted from kind_mod.f90 for names,below
     INTEGER, PARAMETER :: line_size = 132
     INTEGER, PARAMETER :: tag_size = 8


     TYPE, PUBLIC :: snapshot

          !Most variables will be accessed through functions
          !rather than directly, so make everyting PRIVATE unless
          !otherwise specified

          !see the comments in LAMMPS (particulary global.f) for
          !more information about these variables.

          PRIVATE

          !simulation cell information

               !global size
               REAL*8 :: xprd, yprd, zprd

               !half box lengths
               REAL*8 :: xprd_half, yprd_half, zprd_half

               !periodicity
               INTEGER :: perflagx, perflagy, perflagz
               INTEGER :: slabflag
```

```
            logical :: nonperiodic


          !counter variables
                    INTEGER :: natoms, nbonds, nangles
                    INTEGER :: ndihedrals, nimpropers

                    INTEGER, allocatable :: numbond(:),
numangle(:)
                    INTEGER, allocatable :: numdihed(:),
numimpro(:)

                    INTEGER, allocatable :: num1bond(:),
num2bond(:)
                    INTEGER, allocatable :: num3bond(:)

                    INTEGER, allocatable :: specbond(:,:)

          !associated arrays for bonds, etc.
                    INTEGER, allocatable :: bondatom1(:,:),
bondatom2(:,:)

                    INTEGER, allocatable :: angleatom1(:,:),
angleatom2(:,:)
                   INTEGER, allocatable :: angleatom3(:,:)

                    INTEGER, allocatable :: dihedatom1(:,:),
dihedatom2(:,:)
                    INTEGER, allocatable :: dihedatom3(:,:),
dihedatom4(:,:)

                    INTEGER, allocatable :: improatom1(:,:),
improatom2(:,:)
                    INTEGER, allocatable :: improatom3(:,:),
improatom4(:,:)

          !type counters
                    INTEGER :: ntypes, nbondtypes, nangletypes
                    INTEGER :: ndihedtypes, nimprotypes

          !informational arrays
                    REAL*8, allocatable :: mass(:), q(:)
                       CHARACTER(tag_size) , DIMENSION (:),
allocatable :: name
                       CHARACTER(tag_size) , DIMENSION (:),
allocatable :: element

                    INTEGER, allocatable :: tag(:), type(:),
molecule(:), true(:)

                    INTEGER, allocatable :: bondtype(:,:),
angletype(:,:)
                    INTEGER, allocatable :: dihedtype(:,:),
improtype(:,:)
```

```
                    INTEGER, allocatable :: localptr(:)

          !simulation arrays
                    REAL*8, allocatable :: x(:,:), v(:,:), f(:,:)

          !neighbor lists

               !to be added later


     !SOCORRO STUFF


          !name for crystal structure
               CHARACTER(line_size) :: crystal_name

          !lattice parameter
               REAL*8 lattice_param

          !lattice vectors
               REAL*8 :: latvec1(3), latvec2(3), latvec3(3)

          !type of lattice structure
               CHARACTER(line_size) :: lattice_type

          !number of Socorro atoms
               INTEGER num_atoms_soc

          !Socorro atoms
               CHARACTER(tag_size) , DIMENSION (:), allocatable ::
atoms_element_soc
                         INTEGER, allocatable :: atoms_type_soc(:)
               REAL*8, allocatable :: atoms_soc(:,:)

               REAL*8, allocatable :: soc_v(:,:)
               REAL*8, allocatable :: soc_f(:,:)

     !QUANTUM FLAG FOR FORCE CALCULATIONS
               LOGICAL, allocatable :: quantum(:)

        END TYPE  snapshot

!accessor functions

       PUBLIC :: get_natoms, set_natoms
       PUBLIC :: get_nbonds, set_nbonds
       PUBLIC :: get_nangles, set_nangles
       PUBLIC :: get_ndihedrals, set_ndihedrals
       PUBLIC :: get_nimpropers, set_nimpropers
       PUBLIC :: get_ntypes, set_ntypes
       PUBLIC :: get_nbondtypes, set_nbondtypes
       PUBLIC :: get_nangletypes, set_nangletypes
       PUBLIC :: get_ndihedtypes, set_ndihedtypes
```

```
        PUBLIC :: get_nimprotypes, set_nimprotypes
        PUBLIC :: get_xprd, set_xprd
        PUBLIC :: get_yprd, set_yprd
        PUBLIC :: get_zprd, set_zprd
        PUBLIC :: get_xprd_half, set_xprd_half
        PUBLIC :: get_yprd_half, set_yprd_half
        PUBLIC :: get_zprd_half, set_zprd_half
        PUBLIC :: get_perflagx, set_perflagx
        PUBLIC :: get_perflagy, set_perflagy
        PUBLIC :: get_perflagz, set_perflagz
        PUBLIC :: get_slabflag, set_slabflag
        PUBLIC :: get_nonperiodic, set_nonperiodic
        PUBLIC :: get_crystal_name, get_lattice_param
        PUBLIC :: get_latvec1, get_latvec2, get_latvec3
        PUBLIC :: get_lattice_type, get_num_atoms_soc
        PUBLIC :: get_atoms_soc, get_atoms_element_soc
        PUBLIC :: get_atoms_type_soc
        PUBLIC :: get_soc_v, get_soc_f
        PUBLIC :: set_crystal_name, set_lattice_param
        PUBLIC :: set_latvec1, set_latvec2, set_latvec3
        PUBLIC :: set_lattice_type, set_num_atoms_soc
        PUBLIC :: set_atoms_soc, set_atoms_element_soc
        PUBLIC :: set_atoms_type_soc
        PUBLIC :: set_soc_v, set_soc_f
        PUBLIC :: alloc_mass, get_mass, set_mass
        PUBLIC :: alloc_element, get_element, set_element
        PUBLIC :: alloc_name, get_name, set_name
        PUBLIC :: alloc_q, get_q, set_q
        PUBLIC :: alloc_x, get_x, set_x
        PUBLIC :: get_x_L2S, set_x_S2L
        PUBLIC :: alloc_v, get_v, set_v
        PUBLIC :: alloc_f, get_f, set_f
        PUBLIC :: set_f_S2L
        PUBLIC :: alloc_tag, get_tag, set_tag
        PUBLIC :: alloc_type, get_type, set_type
        PUBLIC :: alloc_molecule, get_molecule, set_molecule
        PUBLIC :: alloc_true, get_true, set_true
        PUBLIC :: alloc_atoms_soc,alloc_atoms_element_soc
        PUBLIC :: alloc_atoms_type_soc
        PUBLIC :: alloc_soc_v,alloc_soc_f
        PUBLIC :: get_quantum, set_quantum, alloc_quantum


        CONTAINS

!SET-GET-SET-GET-SET-GET-SET-GET-SET-GET-SET-GET-SET-GET-SET-GET

     FUNCTION get_natoms(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_natoms

          get_natoms = structure%natoms
```

```
     END FUNCTION get_natoms


     SUBROUTINE set_natoms(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%natoms = n

     END SUBROUTINE set_natoms

!---------------------

     FUNCTION get_nbonds(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_nbonds

          get_nbonds = structure%nbonds

     END FUNCTION get_nbonds


     SUBROUTINE set_nbonds(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%nbonds = n

     END SUBROUTINE set_nbonds

!---------------------

     FUNCTION get_nangles(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_nangles

          get_nangles = structure%nangles

     END FUNCTION get_nangles


     SUBROUTINE set_nangles(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%nangles = n

     END SUBROUTINE set_nangles
```

```
!---------------------

     FUNCTION get_ndihedrals(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_ndihedrals

          get_ndihedrals = structure%ndihedrals

     END FUNCTION get_ndihedrals


     SUBROUTINE set_ndihedrals(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%ndihedrals = n

     END SUBROUTINE set_ndihedrals

!---------------------

     FUNCTION get_nimpropers(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_nimpropers

          get_nimpropers = structure%nimpropers

     END FUNCTION get_nimpropers


     SUBROUTINE set_nimpropers(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%nimpropers = n

     END SUBROUTINE set_nimpropers

!---------------------

     FUNCTION get_ntypes(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_ntypes

          get_ntypes = structure%ntypes

     END FUNCTION get_ntypes
```

```
     SUBROUTINE set_ntypes(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%ntypes = n

     END SUBROUTINE set_ntypes

!---------------------

     FUNCTION get_nbondtypes(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_nbondtypes

          get_nbondtypes = structure%nbondtypes

     END FUNCTION get_nbondtypes


     SUBROUTINE set_nbondtypes(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%nbondtypes = n

     END SUBROUTINE set_nbondtypes

!---------------------

     FUNCTION get_nangletypes(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_nangletypes

          get_nangletypes = structure%nangletypes

     END FUNCTION get_nangletypes


     SUBROUTINE set_nangletypes(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%nangletypes = n

     END SUBROUTINE set_nangletypes

!---------------------

     FUNCTION get_ndihedtypes(structure)
```

```
          TYPE (snapshot) :: structure
          INTEGER :: get_ndihedtypes

          get_ndihedtypes = structure%ndihedtypes

     END FUNCTION get_ndihedtypes


     SUBROUTINE set_ndihedtypes(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%ndihedtypes = n

     END SUBROUTINE set_ndihedtypes

!---------------------

     FUNCTION get_nimprotypes(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_nimprotypes

          get_nimprotypes = structure%nimprotypes

     END FUNCTION get_nimprotypes


     SUBROUTINE set_nimprotypes(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%nimprotypes = n

     END SUBROUTINE set_nimprotypes

!---------------------

     FUNCTION get_xprd(structure)

          TYPE (snapshot) :: structure
          REAL*8 :: get_xprd

          get_xprd = structure%xprd

     END FUNCTION get_xprd


     SUBROUTINE set_xprd(structure,n)

          TYPE (snapshot) :: structure
```

```
          REAL*8, INTENT(IN):: n

          structure%xprd = n

     END SUBROUTINE set_xprd

!---------------------

     FUNCTION get_yprd(structure)

          TYPE (snapshot) :: structure
          REAL*8 :: get_yprd

          get_yprd = structure%yprd

     END FUNCTION get_yprd


     SUBROUTINE set_yprd(structure,n)

          TYPE (snapshot) :: structure
          REAL*8, INTENT(IN):: n

          structure%yprd = n

     END SUBROUTINE set_yprd

!---------------------

     FUNCTION get_zprd(structure)

          TYPE (snapshot) :: structure
          REAL*8 :: get_zprd

          get_zprd = structure%zprd

     END FUNCTION get_zprd


     SUBROUTINE set_zprd(structure,n)

          TYPE (snapshot) :: structure
          REAL*8, INTENT(IN):: n

          structure%zprd = n

     END SUBROUTINE set_zprd

!---------------------

     FUNCTION get_xprd_half(structure)

          TYPE (snapshot) :: structure
          REAL*8 :: get_xprd_half
```

```
          get_xprd_half = structure%xprd_half

     END FUNCTION get_xprd_half


     SUBROUTINE set_xprd_half(structure,n)

          TYPE (snapshot) :: structure
          REAL*8, INTENT(IN):: n

          structure%xprd_half = n

     END SUBROUTINE set_xprd_half

!---------------------

     FUNCTION get_yprd_half(structure)

          TYPE (snapshot) :: structure
          REAL*8 :: get_yprd_half

          get_yprd_half = structure%yprd_half

     END FUNCTION get_yprd_half


     SUBROUTINE set_yprd_half(structure,n)

          TYPE (snapshot) :: structure
          REAL*8, INTENT(IN):: n

          structure%yprd_half = n

     END SUBROUTINE set_yprd_half

!---------------------

     FUNCTION get_zprd_half(structure)

          TYPE (snapshot) :: structure
          REAL*8 :: get_zprd_half

          get_zprd_half = structure%zprd_half

     END FUNCTION get_zprd_half


     SUBROUTINE set_zprd_half(structure,n)

          TYPE (snapshot) :: structure
          REAL*8, INTENT(IN):: n

          structure%zprd_half = n
```

```
     END SUBROUTINE set_zprd_half

!---------------------

     FUNCTION get_perflagx(structure)

          TYPE (snapshot) :: structure
          INTEGER:: get_perflagx

          get_perflagx = structure%perflagx

     END FUNCTION get_perflagx


     SUBROUTINE set_perflagx(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%perflagx = n

     END SUBROUTINE set_perflagx

!---------------------

     FUNCTION get_perflagy(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_perflagy

          get_perflagy = structure%perflagy

     END FUNCTION get_perflagy


     SUBROUTINE set_perflagy(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%perflagy = n

     END SUBROUTINE set_perflagy

!---------------------

     FUNCTION get_perflagz(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_perflagz

          get_perflagz = structure%perflagz
```

```
     END FUNCTION get_perflagz


     SUBROUTINE set_perflagz(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%perflagz = n

     END SUBROUTINE set_perflagz

!---------------------

     FUNCTION get_slabflag(structure)

          TYPE (snapshot) :: structure
          INTEGER :: get_slabflag

          get_slabflag = structure%slabflag

     END FUNCTION get_slabflag


     SUBROUTINE set_slabflag(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%slabflag = n

     END SUBROUTINE set_slabflag

!---------------------

     FUNCTION get_nonperiodic(structure)

          TYPE (snapshot) :: structure
          LOGICAL :: get_nonperiodic

          get_nonperiodic = structure%nonperiodic

     END FUNCTION get_nonperiodic


     SUBROUTINE set_nonperiodic(structure,n)

          TYPE (snapshot) :: structure
          LOGICAL, INTENT(IN):: n

          structure%nonperiodic = n

     END SUBROUTINE set_nonperiodic
```

```
!---------------------

     SUBROUTINE alloc_mass(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          allocate(structure%mass(n))

     END SUBROUTINE alloc_mass


     FUNCTION get_mass(structure,n)

          TYPE (snapshot) :: structure
          REAL*8 :: get_mass
          INTEGER, INTENT(IN):: n

          get_mass = structure%mass(n)

     END FUNCTION get_mass


     SUBROUTINE set_mass(structure,n,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          REAL*8, INTENT(IN):: val

          structure%mass(n) = val

     END SUBROUTINE set_mass

!---------------------

     SUBROUTINE alloc_element(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          allocate(structure%element(n))

     END SUBROUTINE alloc_element


     FUNCTION get_element(structure,n)

          TYPE (snapshot) :: structure
          CHARACTER(8) :: get_element
          INTEGER, INTENT(IN):: n

          get_element = structure%element(n)

     END FUNCTION get_element
```

```
     SUBROUTINE set_element(structure,n,str)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          CHARACTER(tag_size), INTENT(IN):: str

          structure%element(n) = str

     END SUBROUTINE set_element

!---------------------

     SUBROUTINE alloc_name(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          allocate(structure%name(n))

     END SUBROUTINE alloc_name


     FUNCTION get_name(structure,n)

          TYPE (snapshot) :: structure
          CHARACTER(8) :: get_name
          INTEGER, INTENT(IN):: n

          get_name = structure%name(n)

     END FUNCTION get_name


     SUBROUTINE set_name(structure,n,str)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          CHARACTER(tag_size), INTENT(IN):: str

          structure%name(n) = str

     END SUBROUTINE set_name

!---------------------


     SUBROUTINE alloc_q(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          allocate(structure%q(n))
```

```
     END SUBROUTINE alloc_q


     FUNCTION get_q(structure,n)

          TYPE (snapshot) :: structure
          REAL*8 :: get_q
          INTEGER, INTENT(IN):: n

          get_q = structure%q(n)

     END FUNCTION get_q


     SUBROUTINE set_q(structure,n,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          REAL*8, INTENT(IN):: val

          structure%q(n) = val

     END SUBROUTINE set_q

!---------------------

     SUBROUTINE alloc_x(structure,n,m)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n,m

          allocate(structure%x(n,m))

     END SUBROUTINE alloc_x


     FUNCTION get_x(structure,n,m)

          TYPE (snapshot) :: structure
          REAL*8 :: get_x
          INTEGER, INTENT(IN):: n,m

          get_x = structure%x(n,m)

     END FUNCTION get_x

     FUNCTION get_x_L2S(structure,n,m)

          TYPE (snapshot) :: structure
          REAL*8 :: get_x_L2S
          INTEGER, INTENT(IN):: n,m

! convert from LAMMPS units (Angstroms) to Socorro (Bohr)
```

```
          get_x_L2S = structure%x(n,m)*1.88973

     END FUNCTION get_x_L2S


     SUBROUTINE set_x(structure,n,m,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n,m
          REAL*8, INTENT(IN):: val

          structure%x(n,m) = val

     END SUBROUTINE set_x


     SUBROUTINE set_x_S2L(structure,n,m,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n,m
          REAL*8, INTENT(IN):: val

! convert Socorro units (Bohr) to LAMMPS (Angstroms)

          structure%x(n,m) = val*0.529177

     END SUBROUTINE set_x_S2L
!---------------------

     SUBROUTINE alloc_v(structure,n,m)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n,m

          allocate(structure%v(n,m))

     END SUBROUTINE alloc_v


     FUNCTION get_v(structure,n,m)

          TYPE (snapshot) :: structure
          REAL*8 :: get_v
          INTEGER, INTENT(IN):: n,m

          get_v = structure%v(n,m)

     END FUNCTION get_v


     SUBROUTINE set_v(structure,n,m,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n,m
```

```
          REAL*8, INTENT(IN):: val

          structure%v(n,m) = val

     END SUBROUTINE set_v

!---------------------

     SUBROUTINE alloc_f(structure,n,m)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n,m

          allocate(structure%f(n,m))

     END SUBROUTINE alloc_f


     FUNCTION get_f(structure,n,m)

          TYPE (snapshot) :: structure
          REAL*8 :: get_f
          INTEGER, INTENT(IN):: n,m

          get_f = structure%f(n,m)

     END FUNCTION get_f


     SUBROUTINE set_f(structure,n,m,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n,m
          REAL*8, INTENT(IN):: val

          structure%f(n,m) = val

     END SUBROUTINE set_f

     SUBROUTINE set_f_S2L(structure,n,m,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n,m
          REAL*8, INTENT(IN):: val

!Convert from Socorro units (Rydbergs/Bohr) to LAMMPS (
(g/mol)*A^2/fs^2)
          structure%f(n,m) = val*.24807

     END SUBROUTINE set_f_S2L

!---------------------

     SUBROUTINE alloc_tag(structure,n)
```

```
        TYPE (snapshot) :: structure
        INTEGER, INTENT(IN):: n

        allocate(structure%tag(n))

    END SUBROUTINE alloc_tag


    FUNCTION get_tag(structure,n)

        TYPE (snapshot) :: structure
        INTEGER :: get_tag
        INTEGER, INTENT(IN):: n

        get_tag = structure%tag(n)

    END FUNCTION get_tag


    SUBROUTINE set_tag(structure,n,val)

        TYPE (snapshot) :: structure
        INTEGER, INTENT(IN):: n
        INTEGER, INTENT(IN):: val

        structure%tag(n) = val

    END SUBROUTINE set_tag

!---------------------


    SUBROUTINE alloc_type(structure,n)

        TYPE (snapshot) :: structure
        INTEGER, INTENT(IN):: n

        allocate(structure%type(n))

    END SUBROUTINE alloc_type


    FUNCTION get_type(structure,n)

        TYPE (snapshot) :: structure
        INTEGER :: get_type
        INTEGER, INTENT(IN):: n

        get_type = structure%type(n)

    END FUNCTION get_type
```

```
     SUBROUTINE set_type(structure,n,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          INTEGER, INTENT(IN):: val

          structure%type(n) = val

     END SUBROUTINE set_type

!---------------------


     SUBROUTINE alloc_molecule(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          allocate(structure%molecule(n))

     END SUBROUTINE alloc_molecule


     FUNCTION get_molecule(structure,n)

          TYPE (snapshot) :: structure
          INTEGER :: get_molecule
          INTEGER, INTENT(IN):: n

          get_molecule = structure%molecule(n)

     END FUNCTION get_molecule


     SUBROUTINE set_molecule(structure,n,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          INTEGER, INTENT(IN):: val

          structure%molecule(n) = val

     END SUBROUTINE set_molecule

!---------------------


     SUBROUTINE alloc_true(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          allocate(structure%true(n))
```

```
     END SUBROUTINE alloc_true


     FUNCTION get_true(structure,n)

          TYPE (snapshot) :: structure
          INTEGER :: get_true
          INTEGER, INTENT(IN):: n

          get_true = structure%true(n)

     END FUNCTION get_true


     SUBROUTINE set_true(structure,n,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          INTEGER, INTENT(IN):: val

          structure%true(n) = val

     END SUBROUTINE set_true

!---------------------

     FUNCTION get_crystal_name(structure)

          TYPE (snapshot) :: structure
          CHARACTER(132) :: get_crystal_name

          get_crystal_name = structure%crystal_name

     END FUNCTION get_crystal_name


     SUBROUTINE set_crystal_name(structure,name)

          TYPE (snapshot) :: structure
          CHARACTER(*), INTENT(IN):: name

          structure%crystal_name = name

     END SUBROUTINE set_crystal_name

!---------------------

     FUNCTION get_lattice_type(structure)

          TYPE (snapshot) :: structure
          CHARACTER(8) :: get_lattice_type

          get_lattice_type = structure%lattice_type
```

```
     END FUNCTION get_lattice_type


     SUBROUTINE set_lattice_type(structure,name)

          TYPE (snapshot) :: structure
          CHARACTER(*), INTENT(IN):: name

          structure%lattice_type = name

     END SUBROUTINE set_lattice_type

!---------------------

     FUNCTION get_lattice_param(structure)

          TYPE (snapshot) :: structure
          REAL*8 :: get_lattice_param

          get_lattice_param = structure%lattice_param

     END FUNCTION get_lattice_param


     SUBROUTINE set_lattice_param(structure,val)

          TYPE (snapshot) :: structure
          REAL*8, INTENT(IN):: val

          structure%lattice_param = val

     END SUBROUTINE set_lattice_param

!---------------------

     FUNCTION get_latvec1(structure,n)

          TYPE (snapshot) :: structure
          REAL*8 :: get_latvec1
          INTEGER, INTENT(IN):: n

          get_latvec1 = structure%latvec1(n)

     END FUNCTION get_latvec1


     SUBROUTINE set_latvec1(structure,n,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          REAL*8, INTENT(IN):: val

          structure%latvec1(n) = val
```

```
     END SUBROUTINE set_latvec1

!---------------------

     FUNCTION get_latvec2(structure,n)

          TYPE (snapshot) :: structure
          REAL*8 :: get_latvec2
          INTEGER, INTENT(IN):: n

          get_latvec2 = structure%latvec2(n)

     END FUNCTION get_latvec2


     SUBROUTINE set_latvec2(structure,n,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          REAL*8, INTENT(IN):: val

          structure%latvec2(n) = val

     END SUBROUTINE set_latvec2

!---------------------


     FUNCTION get_latvec3(structure,n)

          TYPE (snapshot) :: structure
          REAL*8 :: get_latvec3
          INTEGER, INTENT(IN):: n

          get_latvec3 = structure%latvec3(n)

     END FUNCTION get_latvec3


     SUBROUTINE set_latvec3(structure,n,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          REAL*8, INTENT(IN):: val

          structure%latvec3(n) = val

     END SUBROUTINE set_latvec3

!---------------------

     FUNCTION get_num_atoms_soc(structure)

          TYPE (snapshot) :: structure
```

```
          INTEGER :: get_num_atoms_soc

          get_num_atoms_soc = structure%num_atoms_soc

     END FUNCTION get_num_atoms_soc


     SUBROUTINE set_num_atoms_soc(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          structure%num_atoms_soc = n

     END SUBROUTINE set_num_atoms_soc

!---------------------

     SUBROUTINE alloc_atoms_soc(structure,n,m)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n,m

          allocate(structure%atoms_soc(n,m))

     END SUBROUTINE alloc_atoms_soc


     FUNCTION get_atoms_soc(structure,n,m)

          TYPE (snapshot) :: structure
          REAL*8 :: get_atoms_soc
          INTEGER, INTENT(IN):: n,m

          get_atoms_soc = structure%atoms_soc(n,m)

     END FUNCTION get_atoms_soc


     SUBROUTINE set_atoms_soc(structure,n,m,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n,m
          REAL*8, INTENT(IN):: val

          structure%atoms_soc(n,m) = val

     END SUBROUTINE set_atoms_soc

!---------------------

     SUBROUTINE alloc_soc_v(structure,n,m)

          TYPE (snapshot) :: structure
```

```
        INTEGER, INTENT(IN):: n,m

        allocate(structure%soc_v(n,m))

    END SUBROUTINE alloc_soc_v


    FUNCTION get_soc_v(structure,n,m)

        TYPE (snapshot) :: structure
        REAL*8 :: get_soc_v
        INTEGER, INTENT(IN):: n,m

        get_soc_v = structure%soc_v(n,m)

    END FUNCTION get_soc_v


    SUBROUTINE set_soc_v(structure,n,m,val)

        TYPE (snapshot) :: structure
        INTEGER, INTENT(IN):: n,m
        REAL*8, INTENT(IN):: val

        structure%soc_v(n,m) = val

    END SUBROUTINE set_soc_v

!---------------------

    SUBROUTINE alloc_soc_f(structure,n,m)

        TYPE (snapshot) :: structure
        INTEGER, INTENT(IN):: n,m

        allocate(structure%soc_f(n,m))

    END SUBROUTINE alloc_soc_f


    FUNCTION get_soc_f(structure,n,m)

        TYPE (snapshot) :: structure
        REAL*8 :: get_soc_f
        INTEGER, INTENT(IN):: n,m

        get_soc_f = structure%soc_f(n,m)

    END FUNCTION get_soc_f


    SUBROUTINE set_soc_f(structure,n,m,val)

        TYPE (snapshot) :: structure
```

```
          INTEGER, INTENT(IN):: n,m
          REAL*8, INTENT(IN):: val

          structure%soc_f(n,m) = val

     END SUBROUTINE set_soc_f

!---------------------


     SUBROUTINE alloc_atoms_element_soc(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          allocate(structure%atoms_element_soc(n))

     END SUBROUTINE alloc_atoms_element_soc


     FUNCTION get_atoms_element_soc(structure,n)

          TYPE (snapshot) :: structure
          CHARACTER(8) :: get_atoms_element_soc
          INTEGER, INTENT(IN):: n

          get_atoms_element_soc = structure%atoms_element_soc(n)

     END FUNCTION get_atoms_element_soc


     SUBROUTINE set_atoms_element_soc(structure,n,type)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          CHARACTER(tag_size), INTENT(IN):: type

          structure%atoms_element_soc(n) = type

     END SUBROUTINE set_atoms_element_soc

!---------------------


     SUBROUTINE alloc_atoms_type_soc(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          allocate(structure%atoms_type_soc(n))

     END SUBROUTINE alloc_atoms_type_soc
```

```
     FUNCTION get_atoms_type_soc(structure,n)

          TYPE (snapshot) :: structure
          INTEGER :: get_atoms_type_soc
          INTEGER, INTENT(IN):: n

          get_atoms_type_soc = structure%atoms_type_soc(n)

     END FUNCTION get_atoms_type_soc


     SUBROUTINE set_atoms_type_soc(structure,n,type)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          INTEGER, INTENT(IN):: type

          structure%atoms_type_soc(n) = type

     END SUBROUTINE set_atoms_type_soc

!---------------------

     SUBROUTINE alloc_quantum(structure,n)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n

          allocate(structure%quantum(n))

     END SUBROUTINE alloc_quantum


     FUNCTION get_quantum(structure,n)

          TYPE (snapshot) :: structure
          LOGICAL :: get_quantum
          INTEGER, INTENT(IN):: n

          get_quantum = structure%quantum(n)

     END FUNCTION get_quantum


     SUBROUTINE set_quantum(structure,n,val)

          TYPE (snapshot) :: structure
          INTEGER, INTENT(IN):: n
          LOGICAL, INTENT(IN):: val

          structure%quantum(n) = val

     END SUBROUTINE set_quantum
```

```
!----------------------
!SET-GET-SET-GET-SET-GET-SET-GET-SET-GET-SET-GET-SET-GET-SET-GET

        END MODULE structure_snapshot
```