Simulating the Calcination Process for Boron and Nitrogen Co-Doped Carbon in Lithium-Ion Battery Cathode Materials $LiFePO_4$

(Project CSCI 596)

Ning Nie

1. Introduction

Lithium-ion batteries (LIBs) as a cornerstone technology in modern energy storage systems, powering everything from portable electronics to electric vehicles (EVs) and renewable energy storage solutions. The heart of these batteries lies in their cathode materials, which play a pivotal role in defining the battery's overall performance, energy density, and longevity. However, the efficiency and performance of LIBs are significantly constrained by the electrochemical properties of traditional cathode materials. These materials often exhibit low electronic conductivity and poor lithium-ion (Li^+) diffusion, which are major bottlenecks in achieving high-performance batteries. This limitation is particularly pronounced in the case of lithium iron phosphate ($LiFePO_4$), a widely used cathode material known for its stability and safety but criticized for its intrinsic low electronic conductivity.

In recent years, significant advancements have been made in enhancing the properties of $LiFePO_4$ cathodes. Researchers have explored various strategies, such as nano structuring, surface coating, and doping with foreign atoms, to improve its electrical and ionic conductivity. Among these, doping has shown considerable promise. Doping $LiFePO_4$ with elements like carbon, nitrogen, and sulfur has resulted in improved electronic conductivity and battery performance.

2. Current state of the previous work

My previous work titled "Boron and Nitrogen Co-doped Carbon Layers of $LiFePO_4$ Improve the High-Rate Electrochemical Performance for Lithium-Ion Batteries" discusses an innovative approach to enhancing the electrochemical performance of $LiFePO_4$ used in lithium-ion batteries. This is achieved through the co-doping of nitrogen (N) and boron (B) into carbon layers. My previous research details the preparation, structural and morphological characterizations, and electrochemical measurements of these co-doped materials. The study concludes that single N- or B-doping or N+B co-doping into the carbon layer of $LiFePO_4$ can not only elevate the capacity at high current rates but also create a synergistic effect of nitrogen and boron. This leads to a marked enhancement in the electrochemical performance of both commercial and experimental products. The research demonstrates that $LiFePO_4$ with N+B type co-doped carbon coating is highly effective at high current rates and in retention rates, making it a promising candidate for commercial lithium-ion batteries.

3. Objective

There is a growing recognition of the need for computational simulations in the field of simulating materials preparation. Simulations can play a crucial role in understanding the fundamental mechanisms at the atomic and molecular levels, which often remain elusive in experimental studies. Moreover, computational simulations offer a cost-effective and time-efficient alternative to experimental trial-and-error. They can predict the outcomes of doping, suggest optimal compositions, and even explore new material systems that have not yet been synthesized. This approach is particularly valuable in the context of co-doping strategies, where the interplay between different dopants can lead to a wide range of possible outcomes.

Here in this project, I will focus on bridging the gap: understanding how the nitrogen and boron atoms perform and how their state change during the high temperature calcination process in this doping strategy, the initial research is about the material preparation towards improving the electrochemical performance on the cathode material LiFePO4. See the previous publications on ACS website: (https://pubs.acs.org/doi/10.1021/acsami.5b05398). Given the complexity of the doping process, especially in the context of high-temperature calcination and the insertion of dopants like nitrogen and boron into the carbon layers, there is a critical need for detailed simulations. These simulations can provide insights into how these dopants integrate into the LiFePO4 structure and influence its electrochemical properties.

4. Techniques Proposals

A. Density Functional Theory (DFT) for Electronic Structure Analysis

Density Functional Theory (DFT) is a quantum mechanical method used to investigate the electronic structure of many-body systems. In the context of the co-doped process simulation research, DFT is instrumental in understanding how nitrogen and boron co-doping affects the electronic properties of carbon materials in $LiFePO_4$ cathodes.

To investigate the electronic properties of nitrogen and boron co-doped carbon materials, we implement the algorithm Kohn-Sham equations to solve the electronic density and energy of the system. This involves setting up a supercell model of the doped carbon material and calculating the electronic band structure and density of states. As part of the implementation, we consider utilize software packages like Quantum ESPRESSO or VASP for DFT calculations. These packages allow for the implementation of pseudopotentials and plane-wave basis sets, essential for studying complex materials. Some formula in exchange-correlation functionals, such as the Generalized Gradient Approximation (GGA) or the Perdew-Burke-Ernzerhof (PBE) functional, can be used to accurately model electron interactions.

High-Level Pseudocode for DFT Simulation with MPI and OpenMP:

```
#include <mpi.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

// Constants and global variables
const int numAtoms = 10000; // Number of atoms in the supercell
double latticeParameters[3]; // Lattice parameters for the supercell
double atomicPositions[numAtoms][3]; // Positions of atoms

// Function to set up the supercell model
void setupSupercellModel() {
    // Define lattice parameters, atomic positions, etc.
    // setup might be executed by the master node and then distributed to
worker nodes
    if (world_rank == 0) {
        // Initialize lattice parameters and atomic positions
        // latticeParameters = {a, b, c};
        // atomicPositions = {...}; here will add boron, nitrogen, carbon
atoms mainly
    }
    // Distribute the model data to all nodes
    MPI_Bcast(latticeParameters, 3, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(atomicPositions, numAtoms * 3, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
}

// Function to prepare DFT calculation input
void prepareDFTInput() {
    // Set up pseudopotentials, basis set, exchange-correlation
functional, etc.
    // Could be parallelized if different parts of the system are being
prepared by different processes
    // May define pseudopotentials for each element, according to previous
XPS characterization peak data
    // Parallelize the setup for large systems
#pragma omp parallel
    {
        // Setup pseudopotentials and basis sets for boron, nitrogen,
carbon, and LFP atoms
    }
}

// Function to run DFT calculation
void runDFTCalculation() {
    // Parallel execution of DFT calculations
    // MPI is used to distribute different parts of the calculation across
multiple nodes
    // OpenMP will be used within each node to parallelize calculations at
a finer level
    // Divide the k-point grid among MPI processes
    // Each MPI process handles a subset of k-points here
#pragma omp parallel
    {
        // OpenMP to parallelize calculations for each k-point
```

```
        // Perform DFT calculations (e.g., solving Kohn-Sham equations)
    }
}

// Function to analyze the output
void analyzeOutput() {
    // Analysis of DFT results, which involve gathering data from all
nodes
    // MPI functions can be used to collect and aggregate data from all
processes
    // Gather band structure data (can also be potential, cv, resistance
data) from all nodes
    MPI_Gather(datas); // Gather results from all processes
    if (world_rank == 0) {
        // Master node processes and analyzes the gathered data
    }
}

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    setupSupercellModel();
    prepareDFTInput();

    runDFTCalculation();

    analyzeOutput();

    MPI_Finalize();
    return 0;
}
```

B. Molecular Dynamics (MD) Simulations for Diffusion Analysis

The MD simulations, according to our lecture notes, is a good technique to simulate the movement and interaction of nitrogen and boron atoms within the carbon matrix at high temperatures, providing insights into the doping process during calcination. This part of algorithm is to implement classical Newtonian mechanics to calculate the trajectories of atoms over time. Use appropriate force fields, such as ReaxFF, to model the interactions between carbon, nitrogen, and boron atoms. We can also apply statistical mechanics principles to analyze diffusion coefficients and activation energies, providing a quantitative measure of atom mobility under various temperature conditions. During recent research and industry-wide applications, utilize MD simulation software like LAMMPS or GROMACS. These tools offer a wide range of potentials and algorithms to model atomic interactions accurately.

High-Level Pseudocode for MD Simulation:

```
#include <mpi.h>
#include <omp.h>
#include <cuda_runtime.h>

// Function to calculate forces
void calculateForces(...) {
    // Force calculation logic
}

// CUDA Kernel for force calculations
__global__ void calculateForcesCUDA(...) {
    // CUDA-based force calculation logic
}

// Function to initialize the molecular system
void initializeSystem(...) {
    // Initialize positions, velocities, etc.
    // This mainly contains Nitrogen, Boron, Carbon, and LFP particles
}

// Function to distribute initial conditions to worker nodes
void distributeInitialConditions(...) {
    // Logic to distribute initial conditions to worker nodes
}

// Function to gather results from worker nodes
void gatherResults(...) {
    // Logic to gather results from worker nodes
}

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Differentiate between master and worker nodes
    if (world_rank == 0) {
        // Master node work
        initializeSystem(...);
        distributeInitialConditions(...);
    } else {
        // Worker nodes work
        // Receive initial conditions from master node
        // ...

#pragma omp parallel
        {
            // Parallel force calculations using OpenMP
            calculateForces(...);
        }

        // CUDA kernel call for computationally intensive parts
        calculateForcesCUDA<<<blocks, threads>>>(...);
    }

    // Time integration loop
    for (int step = 0; step < num_steps; step++) {
```

```
        // Update positions and velocities of molecules
        // ...

        // Worker nodes send partial results to master node
        // ...

        if (world_rank == 0) {
            // Master node gathers results from all worker nodes
            gatherResults(...);
        }
    }

    MPI_Finalize();
    return 0;
}
```

C. Thermodynamic and Kinetic Modeling

The thermodynamics and kinetics of the calcination process is essential for optimizing the doping efficiency and material properties. This modeling will provide insights into the reaction rates, activation energies, and phase stability of the doped materials, directly correlating with the findings and objectives of the efficiency and material properties. Employ Arrhenius equations and reaction kinetics models to simulate the calcination process. This involves calculating reaction rates, activation energies, and equilibrium constants. Use thermodynamic data (enthalpy, entropy, Gibbs free energy) to model the phase transformations and stability of the doped materials under different temperature regimes. This part of code implementation will be the custom scripts and software for modeling. Use thermodynamic data (enthalpy, entropy, Gibbs free energy) to model the phase transformations and stability of the doped materials under different temperature regimes.

High-Level Pseudocode for Thermodynamic and Kinetic Modeling:

```
#include <mpi.h>
#include <omp.h>
#include <cuda_runtime.h>
#include <math.h>

// Constants
const double R = 8.314; // Universal gas constant in J/(mol*K)

// CUDA Kernel for intensive calculations
__global__ void intensiveCalculationsCUDA(double* input, double* output,
int dataSize) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < dataSize) {
        output[idx] = dopingOperation(input[idx]);
    }
}

// Device function for intensive operations
__device__ double dopingOperation(double value) {
```

```
    // operations
}

// Calculate reaction rate using Arrhenius equation
double calculateReactionRate(double A, double Ea, double T) {
    double rate = 0.0;
#pragma omp parallel
    {
#pragma omp for reduction(+:rate)
        for (int i = 0; i < some_large_number; ++i) {
            rate += F(Arrhenius)(A, Ea, T, i);
        }
    }
    return rate;
}

// Function to calculate Gibbs free energy
double calculateGibbsFreeEnergy(double enthalpy, double entropy, double T)
{
    double gibbsEnergy = 0.0;
#pragma omp parallel
    {
        gibbsEnergy = enthalpy - T * entropy;
    }
    return gibbsEnergy;
}

// Host function to launch the CUDA kernel
void runIntensiveCalculations(double* hostInput, double* hostOutput, int
dataSize) {
    double *devInput, *devOutput;
    cudaMalloc((void**)&devInput, dataSize * sizeof(double));
    cudaMalloc((void**)&devOutput, dataSize * sizeof(double));
    cudaMemcpy(devInput, hostInput, dataSize * sizeof(double),
cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocks = (dataSize + threadsPerBlock - 1) / threadsPerBlock;
    intensiveCalculationsCUDA<<<blocks, threadsPerBlock>>>(devInput,
devOutput, dataSize);

    cudaMemcpy(hostOutput, devOutput, dataSize * sizeof(double),
cudaMemcpyDeviceToHost);
    cudaFree(devInput);
    cudaFree(devOutput);
}

// Main simulation loop with MPI, OpenMP, and CUDA integration
int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Define parameters for the calcination process
    double A, Ea; // Parameters for Arrhenius equation
    double enthalpy, entropy; // Thermodynamic data
    double T; // Temperature
```

```
    // MPI: Distribute data among nodes
    if (world_rank == 0) {
        // Master node work
    } else {
        // Worker nodes work
    }

    // Calculate reaction rate and Gibbs free energy in parallel
    double reactionRate = calculateReactionRate(A, Ea, T);
    double gibbsFreeEnergy = calculateGibbsFreeEnergy(enthalpy, entropy,
T);

    // Prepare data for CUDA calculations
    double *hostInput, *hostOutput;
    int dataSize = 1000; // the size of data, say 1000 here
    // Allocate and initialize hostInput and hostOutput

    // Run CUDA calculations
    runIntensiveCalculations(hostInput, hostOutput, dataSize);

    // MPI: Gather results from all nodes
    if (world_rank == 0) {
        // Master node gathers results
    }
    // Free host memory
    free(hostInput);
    free(hostOutput);

    MPI_Finalize();
    return 0;
}
```