FastCHGNet: Training one Universal Interatomic Potential to 1.5 Hours with 32 GPUs

Yuanchang Zhou[1,2], Siyu Hu[1,2,*], Chen Wang[1,2], Lin-Wang Wang[2,3], Guangming Tan[1,2], Weile Jia[1,2,*]
[1]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
[2]University of Chinese Academy of Sciences
[3]Institute of Semiconductor, Chinese Academy of Sciences
Email: {zhouyuanchang23s, husiyu, tgm, jiaweile}@ict.ac.cn, wangchen246@mails.ucas.ac.cn, lwwang@semi.ac.cn

***Abstract*—Graph neural network universal interatomic potentials (GNN-UIPs) have demonstrated remarkable generalization and transfer capabilities in material discovery and property prediction. These models can accelerate molecular dynamics (MD) simulation by several orders of magnitude while maintaining *ab initio* accuracy, making them a promising new paradigm in material simulations. One notable example is Crystal Hamiltonian Graph Neural Network (CHGNet), pretrained on the energies, forces, stresses, and magnetic moments from the MPtrj dataset, representing a state-of-the-art GNN-UIP model for charge-informed MD simulations. However, training the CHGNet model is time-consuming(8.3 days on one A100 GPU) for three reasons: (i) requiring multi-layer propagation to reach more distant atom information, (ii) requiring second-order derivatives calculation to finish weights updating and (iii) the implementation of reference CHGNet does not fully leverage the computational capabilities. This paper introduces FastCHGNet, an optimized CHGNet, with three contributions: Firstly, we design innovative Force/Stress Readout modules to decompose Force/Stress prediction. Secondly, we adopt massive optimizations such as kernel fusion, redundancy bypass, etc, to exploit GPU computation power sufficiently. Finally, we extend CHGNet to support multiple GPUs and propose a load-balancing technique to enhance GPU utilization. Numerical results show that FastCHGNet reduces memory footprint by a factor of 3.59. The final training time of FastCHGNet can be decreased to 1.53 hours on 32 GPUs without sacrificing model accuracy.**

***Index Terms*—GNN-UIPs, Molecular dynamics, *ab initio*, GPUs, optimizations.**

I. Introduction

Atomic-level simulations based on Density Functional Theory (DFT) calculations have significantly advanced materials modeling over the past few decades. Recently, the emergence of Universal Interatomic Potentials (UIP) has opened new opportunities for modeling complex materials, such as alloys, amorphous solids, condensed phase liquids, and nanostructured materials [1]–[3]. Unlike dedicated interatomic potentials like DeePMD-kit [4], DTNN [5], SchNet [6], HIP-NN [7], PhysNet [8], or DimeNet [9] that train a separate model for each individual physical system, UIP models are trained on extensive DFT datasets that cover a wide range of elements, aiming to capture the fundamental physics of atomic interactions. Once trained, a UIP model can be applied to diverse physical systems without requiring further DFT calculations. For example, Crystal Hamiltonian Graph Neural Network (CHGNet) [3] and MACE [2], [10] have demonstrated remarkable generalization and transfer capabilities. While maintaining exceptional accuracy, UIP models continue to show great potential in materials science and chemistry, significantly advancing our understanding of atomic interactions and complex materials behavior.

One state-of-the-art UIP model is CHGNet. It is currently the only charge-informed Graph Neural Network(GNN) based interatomic potential (trained with the magnetic moments) and has demonstrated excellent results on various Lithium battery related physical systems. Based on its predicted magnetic moments, CHGNet can accurately represent the orbital occupancy of electrons. It presents strong reliability in predicting properties such as conductivity and activation energies across various structures and compositions (e.g., Li_xFePO_4, $LiMnO_2$) [3].

Despite the remarkable capabilities of CHGNet, its training time is still a significant bottleneck. Training CHGNet on the Materials Project Trajectory Dataset (MPtrj, with 1,580,395 atom configurations) using a single A100 GPU takes about 8.3 days. This extended training time limits the ability to iterate and improve the model quickly, making it difficult for designing new UIP models. We provide an in-depth examination and find that there are three main reasons for the long training time of CHGNet. *First*, the complex model architecture. CHGNet is a GNN-based model and it implements a complex forward pass. Each central atom acquires information from neighboring atoms to update its embedding. By increasing the GNN layers, a central atom can interact with more distant neighbors. For example, in CHGNet, the average number of neighboring atoms increases exponentially with the number of interaction blocks, reaching 104, 10,795, and 1,121,797 for interaction blocks 1 to 3. In practice, the number of interaction blocks often exceeds 3, meaning each atom update can involve over 1 million neighbors. Additionally, the number of bonds and angulars also increases exponentially. This complex model architecture results in heavy computational costs, leading to extensive training times. *Second*, calculating second-order derivatives is necessary for updating model weights. CHGNet predicts energy, force, stress and magnetic moment. According to conservation laws, force is derived by differentiating total energy with respect to atomic positions, while stress is derived

*Corresponding author

by differentiating total energy with respect to the lattice strain tensor. As a result, second-order derivatives are required when using the Adam optimizer to update model weights. The second-order derivatives calculation is a time-consuming process due to the high computational complexity. *Third*, the implementation of reference CHGNet is inefficient. The original CHGNet can only be trained on single GPU with a small minibatch size. This reference implementation contains many serial operations that are not efficiently parallelized. Also, numerous redundant computations that have not been carefully examined and eliminated are included in the implementation of reference CHGNet. It has exhibited high memory usage and large launched CPU kernels in the training. On the whole, there is a lot of room for further optimization and efficiency improvements in the overall training process of CHGNet.

In this paper, we propose FastCHGNet, a highly optimized version of CHGNet. We design a series of strategies to enhance the training efficiency of reference CHGNet. Our major contributions are:

- We propose an innovative module to decouple Energy-Force and Energy-Stress. Meanwhile, we give strict proof to prove that the Force decomposition module meets the rotation equivariant principle.
- We perform efficient parallel optimization strategies such as batching, kernel fusion, redundancy elimination, computational results reuse, etc, to saturate the GPU computation resources.
- We implement an efficient large batch training process by leveraging multi-GPUs. The training parameters are heuristically tuned to ensure a steady and reliable convergence. The distribution of atoms, bonds, and angles is also carefully considered to perform load balance.
- With no sacrifice of accuracy, the training time of CHGNet (8.3 days) can be reduced to 1.53 hours (by using 32 NVIDIA A100 GPUs), gaining a 130x speedup.

II. Background

GNNs have been widely investigated for modeling many-body interactions and have led to a revolution in molecule and atomistic modeling. We will give the general formalization of the message-passing mechanism. This is followed by a rigorous expression of CHGNet workflow.

A. Massage Passing Mechanism

Consider a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ represents the set of vertices and $\mathcal{E}$ represents the set of edges. Each atom i can be viewed as a node and atom feature at layer t denoted by h_i^t. The interaction of atom i and neighbor atom j within a fixed cutoff distance can be regarded as an edge, denoted by e_{ij}. As shown in Eq. 1, GNN propagates information across neighboring edges and aggregates the information into the central atom representation. A non-linear node update operation is applied after the aggregation, where M_t and U_t are learnable messages and node update functions respectively. The central atom can receive the further neighbor atoms' information by stacking GNN interaction blocks.

$$\begin{aligned} \mathbf{m}_i^{t+1} &= \sum_{j \in \mathcal{N}(i)} M_t\left(\mathbf{h}_i^t, \mathbf{h}_j^t, \mathbf{e}_{ij}\right) \\ \mathbf{h}_i^{t+1} &= U_t\left(\mathbf{h}_i^t, \mathbf{m}_i^{t+1}\right) \end{aligned} \tag{1}$$

B. CHGNet

CHGNet is pre-trained on the Materials Project Trajectory Dataset with about 1.5 million inorganic structures under Density Functional Theory (DFT) calculation. The workflow of CHGNet is depicted in Fig. 1(a). CHGNet accepts crystal structures with undetermined atomic charges and predicts corresponding energy, forces, stress, and magnetic moments. Based on the predicted magnetic moments, atomic charges can be inferred under inherent charge constraints. The CHGNet can be divided into 4 parts, Molecular Graph Extraction, Feature Embeddings, Interaction Block, and Output Layer. The details of each part are described as follows.

(1)**Molecular Graph Extraction:** Given a crystal structure with periodic boundary conditions, an atom graph G^a and an auxiliary bond graph G^b can be produced, shown in Fig. 1(b). (a) G^a is used to represent two-body interaction. In G^a, each atom i is treated as a node v_i. Each atom identifies neighboring atoms within its cutoff radius, and the edge e_{ij} connecting node v_i and node v_j is initialized by Cartesian distance $r_{ij} := |r_j - r_i|$. (b) G^b is used to represent three-body interaction. The node in G^b reuses the edge representation in G^a. The edge a_{ijk} in G^b is indicated by angle $\theta_{ijk} = arccos\frac{r_{ij} \cdot r_{ik}}{|r_{ij}| \cdot |r_{ik}|}$ for pairwise information between e_{ij} and e_{ik} in G^a. The number of atoms, bonds, and angles in the crystal structure is written as N_v, N_b and N_a respectively.

(2)**Feature Embedding:** A linear transformation is applied to atomic numbers to get the initialized node feature, denoted as v_i^0. The distance r_{ij} and angular θ_{ijk} are first expanded by trainable smooth Radial Bessel Function(noted as sRBF) [9] and Fourier Transformation(denoted as FT) respectively, and linear transformations($\mathcal{L}(x) = x\mathbf{W} + b$) are then applied to get the bond feature e_{ij}^0, bond weights in atom-conv module e_{ij}^a, bond weights in bond-conv module e_{ij}^b, and angle feature a_{ijk}^0. The transformations are defined as follows:

$$\begin{aligned} v_i^0 &= \mathrm{Z}_i \mathbf{W}_v \colon \mathbb{R}^{N_v \times 1} \to \mathbb{R}^{N_v \times 64} \\ [e_{ij}^0, e_{ij}^a, e_{ij}^b] &= \mathcal{L}\big(\mathrm{sRBF}(r_{ij})\big) \colon \mathbb{R}^{N_v \times 1} \to \mathbb{R}^{2N_b \times (3 \times 64)} \\ a_{ijk}^0 &= \mathcal{L}\big(\mathrm{FT}(\theta_{ijk})\big) \colon \mathbb{R}^{N_a \times 1} \to \mathbb{R}^{N_a \times 64} \end{aligned} \tag{2}$$

(3)**Interaction Block:** The essential parts of CHGNet model are illustrated in Fig. 1(c). The interaction block is in the box with blue background which encodes and updates atoms, bonds, and angles embeddings upon the pre-defined G^a and G^b. The interaction block in the t-th layers:

$$IB^t \colon [v_i^t, e_{ij}^t, a_{ijk}^t, e_{ij}^a, e_{ij}^b] \to [v_i^{t+1}, e_{ij}^{t+1}, a_{ijk}^{t+1}] \tag{3}$$

where $t \in 0, 1, 2$ is the layer of the interaction block. The interaction block contains:

- Atom Conv: node feature and pairwise bond feature are concatenated $f_v = [v_i^t, v_j^t, e_{ij}^t]$ and a GatedMLP ϕ_v^t is

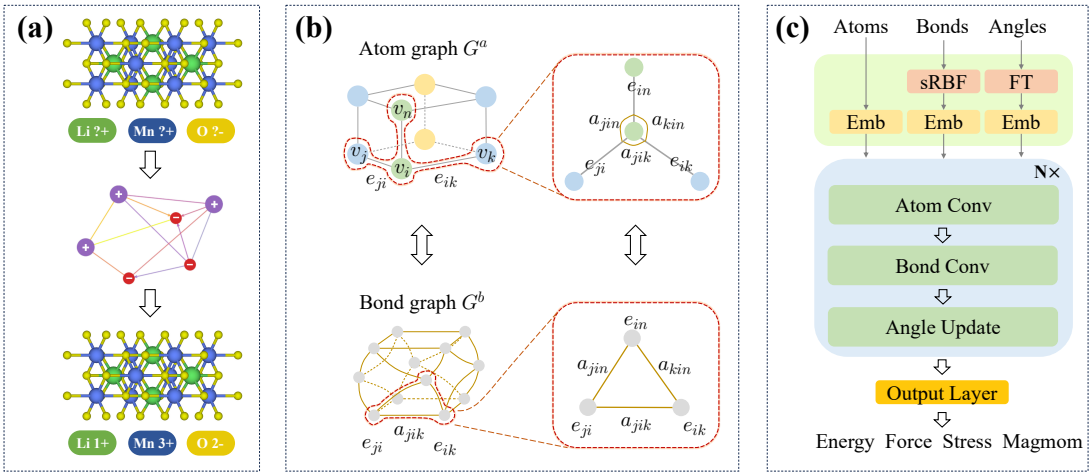

Fig. 1: The CHGNet workflow. (a) The input and output of CHGNet; (b)The graph representation of crystal structure; (c) The core module of CHGNet.

performed on f_v. The weighted message is constructed and aggregated in the following form:

$$v_i^{t+1} = v_i^t + \mathcal{L}_v^t[\sum_{j \in N(i)} e_{ij}^a \odot \phi_v^t(f_v)] \tag{4}$$

- Bond Conv: the input of the bond convolution module is $f_e = [v_i^{t+1}, e_{ij}^t, e_{ik}^t, a_{ijk}^t]$ and a GatedMLP ϕ_e^t is applied on f_e. The bond feature is updated by:

$$e_{ij}^{t+1} = e_{ij}^t + \mathcal{L}_e^t[\sum_{k \in N(i), k \neq j} e_{ij}^b \odot e_{ik}^b \odot \phi_e^t(f_e)] \tag{5}$$

- Angle Update: The input of the Angle Update module is the renewed node, and bond feature, combined with the stale angle features, $f_a = [v_i^{t+1}, e_{ij}^{t+1}, e_{ik}^{t+1}, a_{ijk}^t]$. The angle feature is calculated by:

$$a_{ijk}^{t+1} = a_{ijk}^t + \phi_a^t(f_a) \tag{6}$$

where GatedMLP operation [11] is formulated by $\phi(x) = (\sigma \circ \mathrm{LN} \circ \mathrm{Fc}(x)) \odot (\mathrm{g} \circ \mathrm{LN} \circ \mathrm{Fc}(x))$. σ and g are Sigmoid and SiLu activations. LN and Fc are represented by layer normalization and fully connected operation. $\odot$ is elementwise multiplication.

(4)**Output Layer:** The total energy is derived by summing up the nonlinear projections of the final atomic features. Forces and stress are calculated by automatically differentiating the energy for atomic coordinates and the lattice strain tensor.

III. Method

The CHGNet introduced in the Background suffers from long training time. We develop a set of strategies to fully exploit the computing power of the heterogeneous architecture. Various optimizations for CHGNet are implemented, including output layer decomposition, kernel fusion, redundancy bypass, load balance, etc. The optimized CHGNet is called FastCHGNet.

A. Overview of FastCHGNet

The structure of FastCHGNet is shown in Fig. 2(a). We will introduce FastCHGNet from top to bottom. The referenced 'sRBF' and 'Fourier' modules contain numerous element-wise operations. We perform aggressive kernel fusion. The optimized modules are denoted as 'Fused-sRBF' and 'Fused-Fourier' modules. The Feature Embedding module remains unchanged. The Interaction Block of FastCHGNet is much more efficient than the original Interaction Block because of the dependency elimination of the bond features and the angle features. Besides, multiple scattered small kernels have been fused and the redundant computation has been removed in the Interaction Block. In the prediction of Energy/Force/Stress, we design a novel multi-head module to decouple these properties instead of calculating Force/Stress through the rigorous derivative.

We categorize the optimizations into two major classes, 'Model innovation' and 'System optimizations'. The multi-head decoupling and dependency elimination are introduced in the 'Model innovation'. The kernel fusion, load balance, etc are detailed in 'System Optimizations'.

B. Model innovation

Multi-Head Decomposition: In the reference CHGNet, the Force and Stress are calculated by $F_i = -\frac{\partial E_{\text{tot}}}{\partial x_i}$ and $\sigma = \frac{1}{V}\frac{\partial E_{\text{tot}}}{\partial \epsilon}$, under the guidance of physical conservative rule. The first-order derivatives are required to calculate Force and Stress. The second-order derivatives are needed to update weights. Note that second-order derivatives calculation has higher computational complexity which is storage and computation-intensive.

However, this physical rule is proven not compulsory in tasks such as the initial structure to relaxed energy (IS2RE). Predicting F_i directly can yield better fitting results [12]. In FastCHGNet, we design a sound Force Head and a Stress Head to fit Force and Stress directly. We provide rigorous proof to

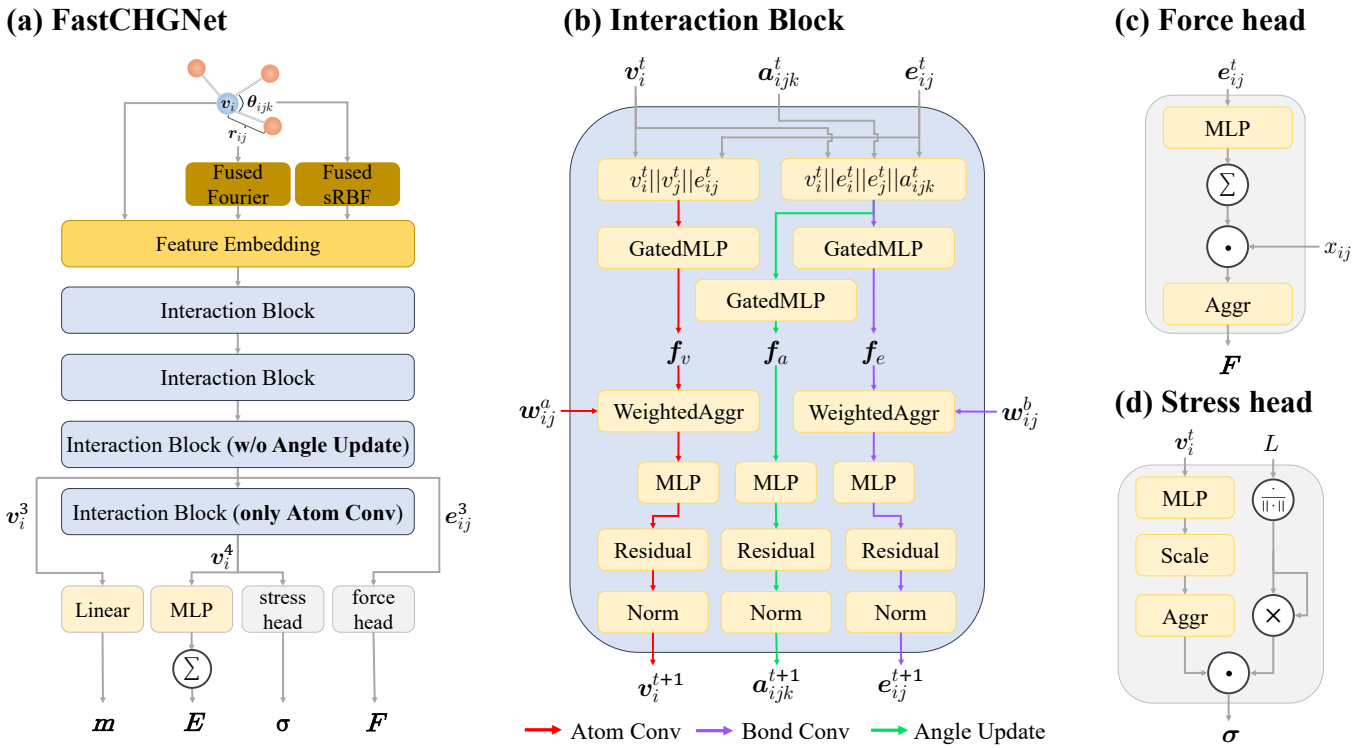

Fig. 2: The architecture of FastCHGNet. (a) the high-level workflow of FastCHGNet; (b) the detailed implementation of Interaction Block; (c) Force head module; (d) Stress head module.

claim that our designed Force decomposition module assures an important property: rotation equivariance.

Force Head: The Force Head in FastCHGNet is shown in Fig. 2(c). This head directly predicts the force using the final bond features e_{ij}^t combined with the bond vectors x_{ij}. It is defined as follows:

$$\begin{aligned} n_{ij} &= MLP(e_{ij}^t) \\ F_i &= \sum_j (n_{ij} \odot x_{ij}) \end{aligned} \tag{7}$$

where n_{ij} and x_{ij} represent the magnitude and direction of the force exerted by atom j on atom i respectively. The net force F_i is the sum of the forces exerted on i by all neighboring atoms j, which is represented by $n_{ij} \odot x_{ij}$. For any rotation matrix R, the bond features e_{ij}^t are invariant and the bond vectors x_{ij} is transformed into Rx_{ij}. Thus, the Force head is rotation equivariant as shown below:

$$\begin{aligned} F_i' &= \sum_j (n_{ij} \odot Rx_{ij}) \\ &= R \sum_j (n_{ij} \odot x_{ij}) \\ &= RF_i \end{aligned} \tag{8}$$

Stress Head: Similarly, we compute the lattice's cross-product to obtain the normal vector of the lattice, and combine it with the atomic representation v_i^t from the final layer to calculate the system's stress. The Stress Head is defined as follows:

$$\sigma = \sum_i ((scale * MLP(v_i^t)) \odot (\sum_{ij} \frac{L_i}{||L_i||} \otimes \frac{L_j}{||L_j||})) \tag{9}$$

where v_i^t is the last layer atom feature and L is lattice vectors. $\otimes$ denotes outer products. The structure is depicted in Fig. 2(d).

Dependency Elimination: In CHGNet, message construction is based on atom, bond, and angle features, which are updated in each Interaction Block. In the Atom Convolution, messages are derived from the concatenated feature vectors of an atom and a bond, specifically v_i^t and e_{ij}^t. For the Bond Convolution, messages come from the updated atomic features v_i^{t+1}, the bond features e_{ij}^t and the angle features a_{ijk}^t. In the Angle Update layer, the messages used to update the angle representation are obtained from the updated atomic features v_i^{t+1} and the bond features e_{ij}^{t+1}, along with the angle features a_{ijk}^t. In the reference Interaction Block, the dependencies are as follows:

$$\begin{aligned} v_i^{t+1} &= \text{Atom Conv}(v_i^t, e_{ij}^t) \\ e_{ij}^{t+1} &= \text{Bond Conv}(v_i^{t+1}, e_{ij}^t, a_{ijk}^t) \\ a_i^{t+1} &= \text{Angle Update}(v_i^{t+1}, e_{ij}^{t+1}, a_{ijk}^t) \end{aligned} \tag{10}$$

In FastCHGNet, as shown in Fig. 2(b), we break the dependency of e_{ij}^{t+1} and a_i^{t+1}, directly utilizing the feature vectors v_i^t, e_{ij}^t to construct the messages in the bond convolution

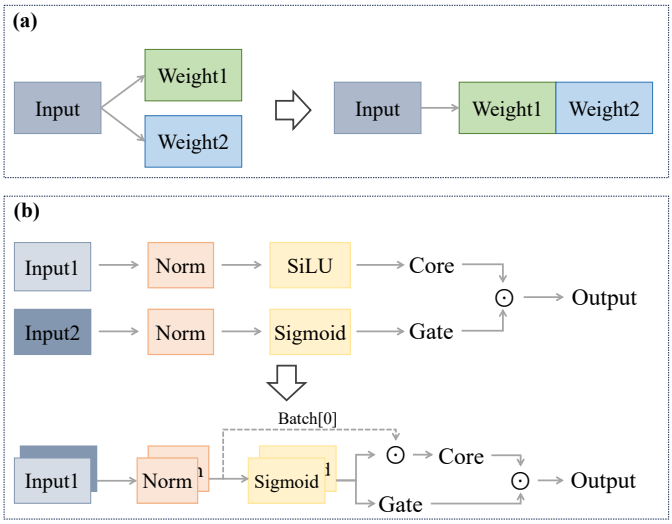

Fig. 3: Packing strategy. (a) Sharing the same input can be fused into a larger matrix-matrix multiplication; (b) The core branch and gate branch fusion strategy of GatedMLP operation.

layer and angle update layer. The dependency of Atom Conv module, Bond Conv module, and Angle Update module is eliminated. The inputs for Bond Conv module and Angle Update module become consistent. Both of the inputs require going through the GatedMLP operation.

$$\begin{aligned} e_{ij}^{t+1} &= \text{Bond Conv}(v_i^t, e_{ij}^t, a_{ijk}^t) \\ a_i^{t+1} &= \text{Angle Update}(v_i^t, e_{ij}^t, a_{ijk}^t) \end{aligned} \tag{11}$$

This modification does not affect accuracy while allowing the forward pass for atoms, bonds, and angles to update concurrently.

C. System optimizations

Parallel Computation of Basis: As depicted in Alg. 1, CHGNet implements the serial computation of $\tilde{e}_{ij}^a$, $\tilde{e}_{ij}^b$, $\tilde{a}_{ijk}$, for each sample in one batch. It iterates through each graph g to extract the lattice $L \in \mathbb{R}^{3\times3}$ and the neighbor image $I \in \mathbb{R}^{n\times3}$, subsequently computing r_{card} along with the Cartesian coordinates of the starting atom r_i and the destination atom r_j. '@' denotes matrix multiplication. The algorithm then employs the smooth radial basis function $sRBF$ and fourier transformation FT to compute $g\_\tilde{e}_{ij}^a$, $g\_\tilde{e}_{ij}^b$ and $g\_\tilde{a}_{ijk}$, appending these values to their respective lists. Ultimately, the results are concatenated along dimension 0 and returned. However, a significant limitation of this algorithm is its reliance on a serial processing approach, which restricts the effective utilization of the parallel capabilities of modern computational hardware. This sequential computation incurs considerable CPU overhead, especially when managing large batch sizes, ultimately resulting in decreased overall performance.

We improved the computational process to maximize GPU utilization. As illustrated in Alg. 2, from lines 2 to 9, we first extract the lattice L, neighbor image I, and r_{card} for each

Algorithm 1 Serial Computation of Basis in Batches

Input: $Crystals$ # all samples in a batch
Output: $\tilde{e}_{ij}^a, \tilde{e}_{ij}^b, \tilde{a}_{ijk}$
1: Initialize empty list $\tilde{e}_{ij}^a, \tilde{e}_{ij}^b, \tilde{a}_{ijk}$
2: **for** each $g \in Crystals$ **do**
3: $L \leftarrow g.lattice$
4: $I \leftarrow g.neighbor\_image$
5: $r_{card} = g.r_{frac}@L$
6: get r_i, r_j from r_{card}
7: $r_j = r_j + I@L$
8: $r_{ij} = r_i - r_j$
9: $g\_\tilde{e}_{ij}^a, g\_\tilde{e}_{ij}^b = sRBF(r_{ij})$
10: append $g\_\tilde{e}_{ij}^a$ to $\tilde{e}_{ij}^a$
11: append $g\_\tilde{e}_{ij}^b$ to $\tilde{e}_{ij}^b$
12: **if** $angle\_nums \neq 0$ **then**
13: caculate θ_{ijk} from r_{ij}
14: $g\_\tilde{a}_{ijk} = FT(\theta_{ijk})$
15: append $g\_\tilde{a}_{ijk}$ to $\tilde{a}_{ijk}$
16: **end if**
17: **end for**
18: Concatenate $\tilde{e}_{ij}^a, \tilde{e}_{ij}^b, \tilde{a}_{ijk}$ along dimension 0
19: **return** $\tilde{e}_{ij}^a, \tilde{e}_{ij}^b, \tilde{a}_{ijk}$

sample, storing this information in the lists B_r_{card}, B_L, and B_I, respectively. Subsequently, we concatenate B_r_{card} and B_L along dimension 0, while B_I is assembled as a block diagonal matrix. From lines 12 to 19, we process all samples in one batch in parallel. By integrating all samples before performing the calculations, we significantly enhance GPU utilization and reduce CPU overhead.

Redundancy Removal: The reference CHGNet implementation introduces numerous tiny kernels that cannot saturate the GPU's computing resource. There exists a lot of redundancy computations in the computation graph. For example, in sRBF calculation, the polynomial envelope function $u(r_{ij})$ contains redundant computations, as shown in Eq. 12.

$$\begin{aligned} u(r_{ij}) = 1 &- \frac{(p+1)(p+2)}{2}\left(\frac{r_{ij}}{r_{cut}}\right)^p \\ &+ p(p+2)\left(\frac{r_{ij}}{r_{cut}}\right)^{(p+1)} \\ &- \frac{p(p+2)}{2}\left(\frac{r_{ij}}{r_{cut}}\right)^{(p+2)} \end{aligned} \tag{12}$$

By factoring out common terms, we can eliminate these redundant calculations, as shown in Eq. 13.

$$u(r_{ij}) = 1 - \frac{p+2}{2}[(p+1)\xi^p + 2p\xi^{(p+1)} - p\xi^{(p+2)}] \tag{13}$$

where ξ denotes $\frac{r_{ij}}{r_{cut}}$.

Computation Graph Reconstruction: There are many small matrix multiplications (GEMMs) and they can be fused or packed together to increase the parallelism, as they share the same input or have no dependency on each other. For instance, a bundle of linear layers sharing the same input can be fused

Algorithm 2 Parallel Computation of Basis in Batches

Input: $Crystals$
Output: $\tilde{e}^a_{ij}, \tilde{e}^b_{ij}, \tilde{a}_{ijk}$

Initialize empty list B_r_{card}, B_L, B_I
for each $g \in Crystals$ **do**
  $L \leftarrow g.lattice$
  $I \leftarrow g.neighbor\_image$
  $r_{card} = g.r_{frac}@L$
  append I to B_I
  append L to B_L
  append r_{card} to B_r_{card}
end for
Concatenate B_r_{card}, B_L along dimension 0
Concatenate B_I as block diagonal matrix
get B_r_i, B_r_j from B_r_{card}
$B\_r_j = B\_r_j + B\_I@B\_L$
$B\_r_{ij} = B\_r_i - B\_r_j$
$\tilde{e}^a_{ij}, \tilde{e}^b_{ij} = sRBF(B\_r_{ij})$
if $angle\_nums \neq 0$ **then**
  caculate B_θ_{ijk} from B_r_{ij}
  $\tilde{a}_{ijk} = FT(B\_\theta_{ijk})$
end if
return $\tilde{e}^a_{ij}, \tilde{e}^b_{ij}, \tilde{a}_{ijk}$

into a larger linear layer by weights concatenation, as depicted in Fig. 3(a). In GatedMLP operation, the same computations such as layer normalization $LN()$, Sigmoid activations, etc are contained in two channels that can be merged, as shown in Fig. 3(b). The $sigmoid(x)$ calculation can be merged due to $silu(x) = x \cdot sigmoid(x)$. The $silu(x)$ result can be derived by multiplying $x(batch\,[0])$, as the dashed line shows in Fig. 3(b). Computation Graph Reconstruction is a critical technique in deep learning applications that combines multi-kernel into one efficient kernel, thereby reducing multiple kernel launches and data transfer overhead.

Load Balance Sampler: The reference CHGNet implementation does not fully exploit the parallelism in the model's computation graph. CHGNet can be trained only on a single GPU. We support multi-GPU training via data parallelism. As the multi-GPU training is employed, the batch size can be greatly increased. In this situation, the load imbalance problem is obvious in multi-GPU training between the samples in a batch as the number of atoms, bonds, and angles of different molecules varies substantially. This imbalance results in a large synchronization overhead in the multi-GPU setting. Fig. 5 describes the distribution of training samples, including the number of nodes, bonds, and angles and their frequency in the MPTrj dataset. It can be easily recognized that the frequency follows a long-tail distribution. To solve the load imbalance problem, we first calculate the total number of atoms, bonds, and angles for each sample in the global batch and sort them in ascending order. Then, each GPU selects the smallest and largest samples from the remaining samples in turn, until all samples are allocated. When the number of GPUs is 4, the sample allocation scheme is shown in Fig. 4. This approach ensures that the computational load for each instance is as balanced as possible.

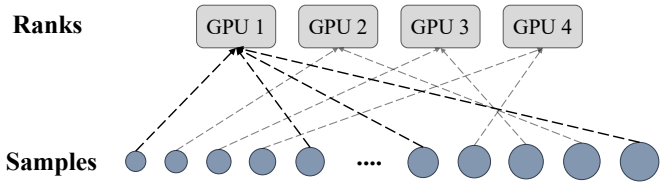

Fig. 4: Load balance sampler. The strategy for distributing samples within a batch across multiple GPUs.

Learning Rate Schedule: When training with large batch sizes, an excessively low learning rate results in insufficient parameter updates, causing the model to move too slowly in the parameter space and ultimately affecting its accuracy. To fully leverage the advantages of large batches, we employ an adaptive learning rate adjustment strategy as follows:

$$init_{LR} = \frac{batchsize}{k} \times 0.0003 \tag{14}$$

where k is a hyper-parameter and we set k as 128. This approach adjusts the learning rate in proportion to the batch size, ensuring a steady and reliable convergence.

Other Optimization:

Data Prefetch: This technique is employed to enhance the efficiency of batches loading during the training process. While the current batch is being processed, data prefetch asynchronously transfers the next mini-batch from CPU memory to GPU memory, reducing data wait times and optimizing computational resource usage. Specifically, this strategy utilizes separate streams for data transfer, allowing the copy operation to occur concurrently with the forward and backward passes.

Communication Overlap: After each backward pass, a global all-reduce operation is required to obtain the gradients. This communication overhead is unavoidable. However, instead of waiting for all gradients calculations to finish, we can perform all-reduce once after the gradient calculation of a part of parameters is completed while the other part of gradients are being calculated. This overlap of communication and calculation minimizes idle time for each compute node and accelerates the overall training process.

IV. Experiment Setup

Hardware and software stacks: All numerical tests are conducted using the GPU cluster. Each node is equipped with two 64-core Intel Xeon Platinum 8358 CPUs and 8 NVIDIA A100 GPUs. Each node has 1TB host memory and each A100 GPU has a memory capacity of 80GB. The CPU bandwidth is 11.2 GT/s and the GPU bandwidth is 1935 GB/s. Node communication operates in a non-blocking fat-tree topology. GCC 11.3.0 is chosen for compiling CPU code and CUDA 12.2 is our GPU compiler. The code is developed on the PyTorch 2.3.1 deep learning platform. The packages pymatgen, ase, etc are also used.

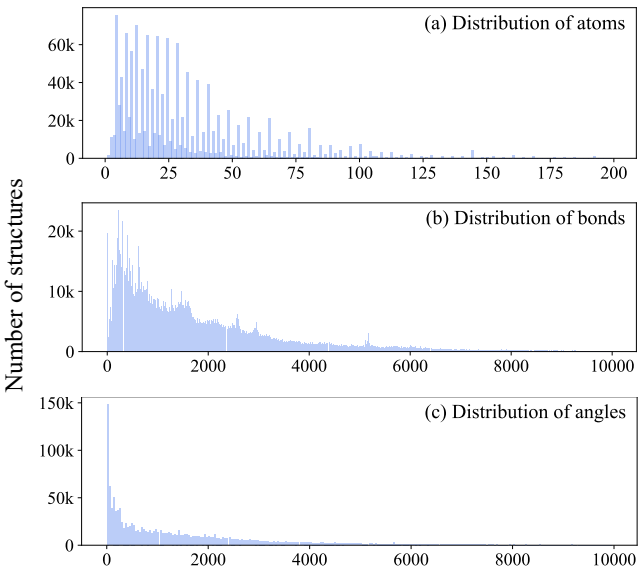

Fig. 5: The atom/bond/angle distribution of MPtrj dataset.

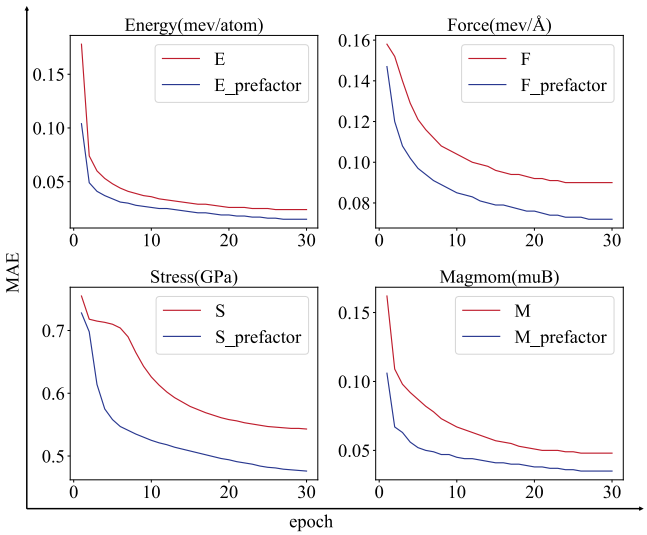

Fig. 6: The convergence of fine-tuned learning rate in terms of Energy, Force, Stress, and Magmom. The red line means the default learning rate. The blue line means the learning rate is multiplied by a prefactor.

Dataset: The MPtrj dataset consists of 1,580,395 inorganic crystal structures composed of 89 elements, with data including 1,580,395 energies, 7,944,833 magnetic moments, 49,295,660 forces, and 14,223,555 stresses, all calculated using DFT. These structures and labels are extracted from both static and relaxation trajectories obtained through GGA/GGA+U calculations in the Materials Project. In constructing the atom and bond graph, we set the default cutoff distances to 6 Å and 3 Å, respectively. For training FastCHGNet, the dataset is divided into training, validation, and test sets in a 0.9:0.05:0.05 ratio.

Parameters Setting: FastCHGNet has 429,046 trainable parameters. The radial and angular basis number is set to 31. The atom, bond, and angle features are embedded into 64-dimensional vectors. The smoothing coefficient p is set to 8. The model predicts energy, force, stress, and magnetic moment. The loss function in backpropagation is Huber loss, with the prefactor defined as 2, 1.5, 0.1, and 0.1 respectively. 'Adam' optimizer is adopted. The initial learning rate is 0.0003 and the cosine annealing scheduler is applied. The activation functions are Sigmoid and SiLU. We train the model for 30 epochs with a batch size of 128. The reference CHGNet is available at: https://github.com/CederGroupHub/chgnet/tree/main/chgnet/pretrained/0.3.0, v0.3.0.

V. Evaluation

A. Convergence results

Table. I describes the MAE result (the lower the better) of CHGNet and FastCHGNet on the MPtrj testing set. In Energy prediction, FastCHGNet reaches a higher accuracy. Overall, FastCHGNet demonstrates comparable accuracy compared to CHGNet in predicting energy(meV/atom), force(meV/Å), stress(GPa) and magmom(μ_B). FastCHGNet(version 'w/o head') means the output layer is not decoupled. This version achieves a higher precision on the four predictions than the reference CHGNet. This is a result of using larger batch sizes and fine-tuning the learning rate. The parameter of the FastCHGNet(version 'w/o head') is slightly reduced because some unnecessary modules are removed. FastCHGNet(version 'F/S head') means the output layer is decoupled by Force head and Stress head. Thereby the number of parameters is larger than the reference CHGNet. The Force and Stress decomposition modules may result in a small decrease in the precision of force and pressure.

Fig. 6 illustrates the effectiveness of the tuning of the learning rate. We increase the batch size to 2048 and trained for 30 epochs, investigating the convergence of Energy (E), Force (F), Stress (S), and Magnetic Moment (M) where MAE is the evaluation metric. The red curves represent the results using the default learning rate (0.003), where E, F, S, and M converge to 24 meV/atom, 90 meV/Å, 0.543 GPa, and 48 $m\mu_B$, respectively. The blue curves show the results after adjusting the learning rate according to Eq. 14, achieving an improved accuracy of 15 meV/atom, 72 meV/Å, 0.476 GPa, and 35 $m\mu_B$, respectively.

Fig. 7 shows how far the fitting results of CHGNet and FastCHGNet in the testing set deviate from those of DFT (ground truth) in terms of energy and force. The x-axis represents the DFT results, and the y-axis represents the predicted results. The testing snapshots would fall on the solid black line when the predictions of the neural network force field match exactly the DFT results. A more accurate CHGNet/FastCHGNet model is one where the predictions are as close as possible to DFT results, with the data points lying as close as possible to the solid black line. The R^2 is calculated

TABLE I: The Mean Absolute Error(MAE) of CHGNet, FastCHGNet on MPTrj test dataset.

model	version	param	Energy(meV/atom)	Force(meV/Å)	Stress(GPa)	Magmom($m\mu_B$)
CHGNet	v0.3.0	412.5K	29	68	0.314	37
FastCHGNet	w/o head	411.2k	26	62	0.270	35
FastCHGNet	F/S head	429.1K	16	73	0.479	36

and an R^2 value closer to 1 indicates a better fit of the model to the data. FastCHGNet has a higher R^2 than CHGNet in energy but a lower R^2 for force. We randomly selected four systems in the testing dataset and their visualizations are in the lower right corner of each subplot in Fig. 7.

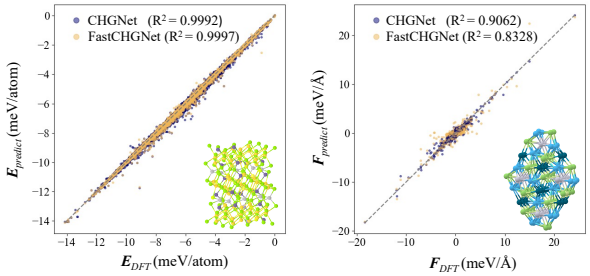

Fig. 7: CHGNet and FastCHGNet performance contrasted with DFT results on the testing dataset.

B. Single GPU

Iteration time & Kernel number: We compare the training performance of CHGNet and FastCHGNet across different batch sizes (16/32/64) on a single NVIDIA Tesla A100 GPU. As shown in Fig. 8(a,b), after a series of optimizations, FastCHGNet achieves a 4.43–5.62× reduction in training time and a 12.72–20.16× decrease in the number of launched kernels. In CHGNet, operations such as 'sRBF' are serial processed and, thus have relatively low computational and resource usage. CHGNet fails to fully leverage the parallelism and resources of GPUs. Therefore, we modified a series of serial computations into parallel computations. And this strategy('Parallel computation of basis') results in a 2.06–2.52× speedup. This improvement is primarily due to the conversion of a series of serial calculations into parallel operations. This approach offers two key benefits: first, alleviating the CPU overhead. FastCHGNet reduces the number of launched kernels from 72659 to 11481 (batch size=64); second, by rewriting the inefficient calculations to an efficient parallel mode, we can enhance GPU utilization. We also employ the 'Kernel fusion + Redundancy bypass' strategy to further reduce the number of launched kernels, resulting in an additional 1.08–1.18× speedup. Furthermore, we implement the decoupling of energy-force and energy-stress calculations by directly computing forces and stresses. The 'decoupling' strategy does not need to compute second-order derivatives in weights updating, resulting in an additional 1.88–2× speedup.

Memory usage: We also compared the GPU memory usage of CHGNet and FastCHGNet under different batch sizes, as shown in Fig. 8(c). GPU memory will slightly increase when 'Parallel computation of basis' is applied. We concatenate certain features to facilitate parallel processing on the GPU, while some features (such as the offset vector) require padding with zeros, leading to increased memory demands. By employing kernel fusion and redundancy removal, we eliminate redundant computations and modules in CHGNet, resulting in a memory reduction of 1.05–1.07×. Furthermore, when we decouple force and stress, the memory usage decreases by a factor of 3.38–3.50×, which is attributed to FastCHGNet's ability to compute force and stress without relying on first-order derivatives, thereby eliminating the need to store intermediate values from the first-order derivative computations.

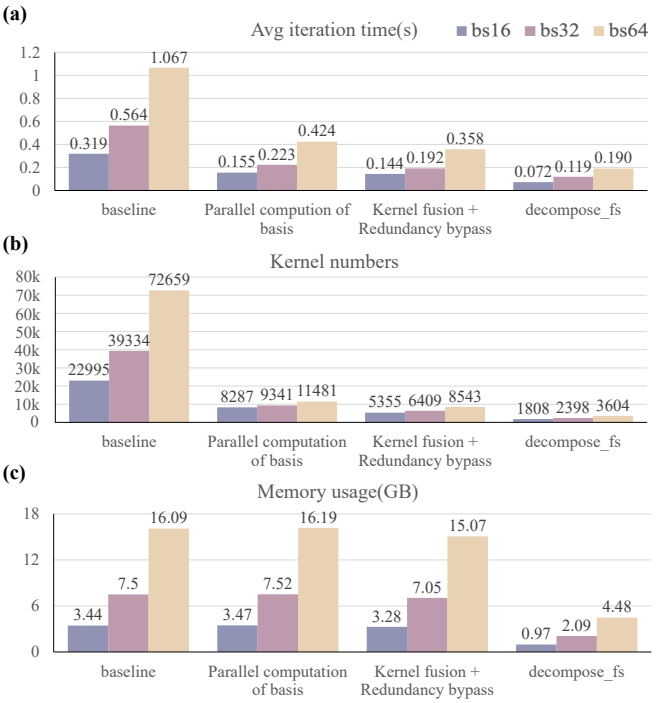

Fig. 8: The average iteration time, the number of launched kernels, and the memory usage under step-by-step system optimization.

C. Multi GPUs

Load Balance: Training on a single A100 GPU still requires 35.4 hours although the optimizations above have significantly improved training speed compared to the reference CHGNet. Training by a larger batch size with multi-GPUs can reduce the absolute training time. With an increase in training batch size, the coefficient of variance(a criterion used to describe

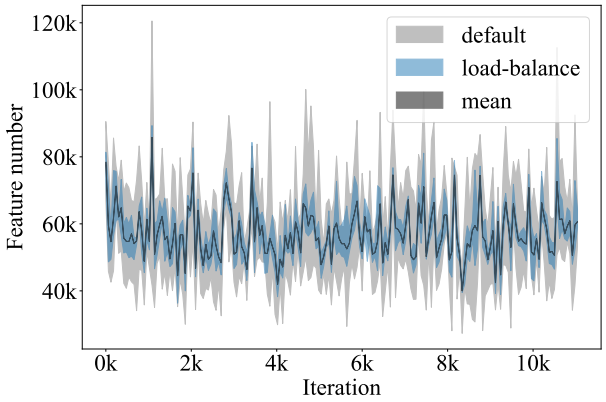

Fig. 9: The feature number of default sampler and load balance sampler.

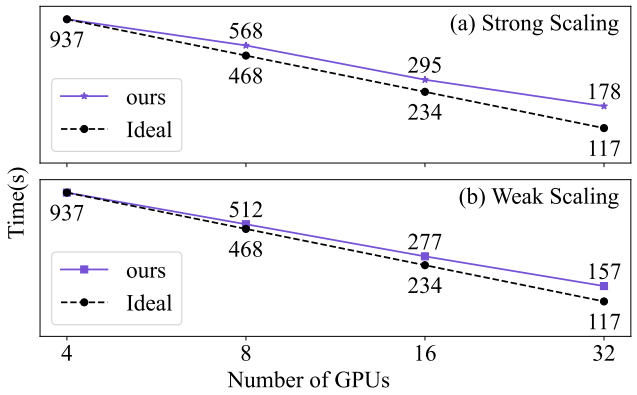

Fig. 10: The strong scaling and weak scaling of FastCHGNet in 4, 8, 16, 32 GPUs.

load imbalance; the higher the variance, the more severe the imbalance) is 0.186(with the default mini-batch size of 32 on 4 GPUs). This indicates that the number of nodes, bonds, and angles varies significantly, as shown in the gray-shaded area of Fig. 9. The x-axis of Fig. 9 is iteration and the y-axis(Feature number) denotes the workload of the training batch. The feature number is the summation of atom number, bond number, and angular number. We design a sampler to address the load imbalance issue, depicted in Section III. By assigning the largest and smallest samples to the same GPU, the coefficient of variance reduces to 0.064 after this strategy, as indicated by the blue-shaded area in Fig. 9.

Strong Scaling: To evaluate the strong scalability of FastCHGNet, we conduct experiments using 4 to 32 GPUs with a global batch size set to 2048. Fig. 10(a) illustrates the strong scalability results. The horizontal axis represents the number of GPUs utilized (note: we use only 4 GPUs for per compute node), while the vertical axis indicates the time required to train one epoch. The speedup is $1.65\times$ when 8 GPUs are utilized with the efficiency 82.5%. When trained by 16 GPUs, we get a $3.18\times$ speedup compared with training by 4 GPUs. The scaling efficiency is 79.5%. The speedup is attributed to the reduced computational workload per GPU. This is because, as the number of GPUs increases, the mini-batch size per GPU decreases. When the training GPUs increase to 32, the training time is reduced by $5.26\times$ compared to training with 4 GPUs. The scaling efficiency is 66%. The lower scaling efficiency comes from the significant communication overhead that arises as the number of GPUs increases.

Weak Scaling: In the weak scaling tests of FastCHGNet, we set the mini-batch size to 512. Testing is performed on 4, 8, 16, and 32 GPUs. The scaling performance is shown in Fig. 10(b). The x-axis of Fig. 10(b) denotes the number of GPUs and the y-axis is the training time of one epoch. The scaling efficiencies for 4, 8, 16, and 32 GPUs are 91.5%, 84.6%, and 74.6%, respectively.

D. Molecular dynamics in real applications

We also compared the performance of CHGNet and FastCHGNet in molecular dynamics. The results are shown in Table. II. We randomly select three systems(LiMnO2, LiTiPO5, Li9Co7O16). They are in different sizes, with feature numbers (the summation of the atoms, bonds, and angles number) of 1088, 3582, and 10188, respectively. The inference speed of FastCHGNet is 2.63 to 3.03 times faster than that of CHGNet. To be more specific, one-step molecular dynamics time before and after optimization is denoted as t and t^*. The speedup $r = \frac{t}{t^*}$ is 2.86, 2.63, and 3.03. The speedup is not as high as the training process. This is due to the given structure is processed step by step in molecular dynamics. One structure molecular dynamics has a low computational cost. In this situation, the GPU computational power cannot be fully utilized.

VI. Related work

Dedicated ML potentials: Machine Learning(ML) methods have been developed to accurately predict atomistic potential energy surfaces (PES) for various applications. There are two main categories of ML potentials. The first category is dedicated ML potentials, which are specifically crafted to describe the PES of a particular system or a very limited class of systems. They require DFT calculations before training, making them impractical for large-scale applications because DFT calculations are expensive [13], [14]. On the other hand, the particular system or the limited class of systems has a relatively low number of degrees of freedom. Thus, dedicated ML potentials are easy to train to converge. The category models tend to be small. The DeePMD [4], BPNN [15], EANN [16], PAINN [17], NequIP [18], NewtonNet [19], GemNet [12], SpookyNet [20], DimeNet++ [21] are dedicated ML potentials.

Universal ML potentials: The second category is universal ML potentials which are developed to simulate entire classes of molecules or crystals. Once universal ML potentials are trained, they can be applied across a broad range of systems

TABLE II: The time required for one-step molecule dynamics of CHGNet, FastCHGNet on LiMnO2, LiTiPO5, Li9Co7O16 structures.

crystal	atoms	bonds	angles	CHGNet	FastCHGNet	speedup
LiMnO2	8	336	744	0.022	0.0077	2.86
LiTiPO5	32	1258	2292	0.021	0.0076	2.63
Li9Co7O16	32	1780	8376	0.023	0.0077	3.03

without further DFT calculations being added. Therefore, universal ML potentials can achieve linear scaling in physical structures. A promising universal model needs a wide range of high-accuracy training data(to provide enough data diversity) and a well-designed model(to describe the potential energy surface). The SevenNet [1], MACE [2], [10], CHGNet [3], M3GNet [22], ALIGNN [23], MEGNet [24], CGCNN [11], GPTFF [25] are universal ML potentials. In this paper, we focus on universal ML potentials. Among these universal ML potentials, CHGNet is the only charge-informed universal ML potential. Note that charge information can provide insights into ionic systems with electronic degrees. Most of the universal ML potentials are GNN backbone for the reason that the message passing scheme is an effective method for learning atomistic systems.

Neural network optimization: Neural network optimization aims to enhance training and inference efficiency, scalability, and performance. Recently, various approaches have been proposed to address the challenges associated with deep neural networks. Among these approaches, deep learning compiler is one of the most prominent tools for improving model efficiency [26], such as TVM [27], TensorRT [28], Ansor [29], XLA [30], etc. They can generate optimized codes automatically. However, they aim at making inferences more efficiently except for XLA. XLA can be used in model training while only supporting Pytorch [31] Framework. In this paper, we focus on model training. Quantization is a popular neural network optimization strategy and has received significant attention recently [32]–[35]. Quantization assigns different bits to different tensors to reduce memory consumption and computation overhead. GNN models can achieve faster inference through quantization. While in the training of machine learning interatomic potentials, quantization has not been successfully applied. The interatomic potential training is sensitive and has extremely high accuracy demand. DeePMD is trained using Float64 and other atomic potential models are trained by Float32. To the best of our knowledge, there are currently no atomic potential models that have been trained using half-precision. Although quantization has been well studied for CNNs, GNNs, and language models, there remains relatively little work utilizing quantization techniques in training atomic potential models. By designing an innovative optimizer, RLEKF, the training process of interatomic potentials can be accelerated [36]. RLEKF converges 8 to 10 times faster [37] than first-order methods on classical networks such as MPT [38], SNAP [39], etc. The parallel quasi-Newton optimizer, FastEKF, demonstrates significant training acceleration for neural network potentials, reducing the training time of DeePMD from days to minutes [40]. EKF-based optimizers provide insights into training MLP-based neural network potentials, while their application to GNN-based potentials requires further exploration. In summary, the training of GNN-based universal ML potentials is still a challenge.

VII. Conclusion

CHGNet is the state-of-the-art GNN-UIP model for charge-informed MD simulations, yet efficiently training this model poses a significant challenge. In this paper, we introduce FastCHGNet, an optimized implementation of CHGNet. The Force and Stress are fitted by the novel Force and Stress head. The memory requirement is greatly reduced. To fully utilize GPU computation resources, a lot of strategies such as kernel fusion, redundancy bypass, etc, have been used to enhance GPU utilization. We also propose a Load Balance Sampler to ensure a relatively even distribution of the computational workload across GPUs. The training time of the reference CHGNet is 8.3 days. Without sacrificing accuracy, the training time of FastCHGNet(without force/stress decoupling) can reduce to 3.79 hours. The training time of a more aggressive version (force/stress will be decoupled) can be reduced to 1.53 hours.

In the future, we plan to design more lightweight modules to replace the currently time-consuming operations, while maintaining model accuracy. In the meantime, we will try to apply model compression and quantization to further accelerate the training process of universal interatomic potentials.

Acknowledgment

This work is supported by the following funding: National Science Foundation of China (92270206, T2125013, 62372435, 62032023, 61972377, 61972380, T2293702), China National Postdoctoral Program for Innovative Talents (BX20240383), CAS Project for Young Scientists in Basic Research (YSBR-005), the Innovation Funding of ICT, CAS under Grant No. E463030. The authors thank the ICT operations team for their strong support.

References

[1] Y. Park, J. Kim, S. Hwang, and S. Han, "Scalable parallel algorithm for graph neural network interatomic potentials in molecular dynamics simulations," Journal of Chemical Theory and Computation, 2024.

[2] I. Batatia, P. Benner, Y. Chiang, A. M. Elena, D. P. Kovács, J. Riebesell, X. R. Advincula, M. Asta, W. J. Baldwin, N. Bernstein et al., "A foundation model for atomistic materials chemistry," arXiv preprint arXiv:2401.00096, 2023.

[3] B. Deng, P. Zhong, K. Jun, J. Riebesell, K. Han, C. J. Bartel, and G. Ceder, "Chgnet as a pretrained universal neural network potential for charge-informed atomistic modelling," Nature Machine Intelligence, vol. 5, no. 9, pp. 1031–1041, 2023.

[4] H. Wang, L. Zhang, J. Han, and E. Weinan, "Deepmd-kit: A deep learning package for many-body potential energy representation and molecular dynamics," Computer Physics Communications, vol. 228, pp. 178–184, 2018.

[5] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," Nature communications, vol. 8, no. 1, p. 13890, 2017.

[6] K. Schütt, P.-J. Kindermans, H. E. Sauceda Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "Schnet: A continuous-filter convolutional neural network for modeling quantum interactions," Advances in neural information processing systems, vol. 30, 2017.

[7] N. Lubbers, J. S. Smith, and K. Barros, "Hierarchical modeling of molecular energies using a deep neural network," The Journal of chemical physics, vol. 148, no. 24, 2018.

[8] O. T. Unke and M. Meuwly, "Physnet: A neural network for predicting energies, forces, dipole moments, and partial charges," Journal of chemical theory and computation, vol. 15, no. 6, pp. 3678–3693, 2019.

[9] J. Gasteiger, J. Groß, and S. Günnemann, "Directional message passing for molecular graphs," in International Conference on Learning Representations (ICLR), 2020.

[10] I. Batatia, D. P. Kovacs, G. Simm, C. Ortner, and G. Csányi, "Mace: Higher order equivariant message passing neural networks for fast and accurate force fields," Advances in Neural Information Processing Systems, vol. 35, pp. 11 423–11 436, 2022.

[11] T. Xie and J. C. Grossman, "Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties," Physical review letters, vol. 120, no. 14, p. 145301, 2018.

[12] J. Gasteiger, F. Becker, and S. Günnemann, "Gemnet: Universal directional graph neural networks for molecules," Advances in Neural Information Processing Systems, vol. 34, pp. 6790–6802, 2021.

[13] J. S. Smith, O. Isayev, and A. E. Roitberg, "Ani-1: an extensible neural network potential with dft accuracy at force field computational cost," Chemical science, vol. 8, no. 4, pp. 3192–3203, 2017.

[14] C. Devereux, J. S. Smith, K. K. Huddleston, K. Barros, R. Zubatyuk, O. Isayev, and A. E. Roitberg, "Extending the applicability of the ani deep learning molecular potential to sulfur and halogens," Journal of Chemical Theory and Computation, vol. 16, no. 7, pp. 4192–4202, 2020.

[15] J. Behler and M. Parrinello, "Generalized neural-network representation of high-dimensional potential-energy surfaces," Physical review letters, vol. 98, no. 14, p. 146401, 2007.

[16] Y. Zhang, C. Hu, and B. Jiang, "Embedded atom neural network potentials: Efficient and accurate machine learning with a physically inspired representation," The journal of physical chemistry letters, vol. 10, no. 17, pp. 4962–4967, 2019.

[17] K. Schütt, O. Unke, and M. Gastegger, "Equivariant message passing for the prediction of tensorial properties and molecular spectra," in International Conference on Machine Learning. PMLR, 2021, pp. 9377–9388.

[18] S. Batzner, A. Musaelian, L. Sun, M. Geiger, J. P. Mailoa, M. Kornbluth, N. Molinari, T. E. Smidt, and B. Kozinsky, "E (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials," Nature communications, vol. 13, no. 1, p. 2453, 2022.

[19] M. Haghighatlari, J. Li, X. Guan, O. Zhang, A. Das, C. J. Stein, F. Heidar-Zadeh, M. Liu, M. Head-Gordon, L. Bertels et al., "Newtonnet: a newtonian message passing network for deep learning of interatomic potentials and forces," Digital Discovery, vol. 1, no. 3, pp. 333–343, 2022.

[20] O. T. Unke, S. Chmiela, M. Gastegger, K. T. Schütt, H. E. Sauceda, and K.-R. Müller, "Spookynet: Learning force fields with electronic degrees of freedom and nonlocal effects," Nature communications, vol. 12, no. 1, p. 7273, 2021.

[21] J. Gasteiger, S. Giri, J. T. Margraf, and S. Günnemann, "Fast and uncertainty-aware directional message passing for non-equilibrium molecules," arXiv preprint arXiv:2011.14115, 2020.

[22] C. Chen and S. P. Ong, "A universal graph deep learning interatomic potential for the periodic table," Nature Computational Science, vol. 2, no. 11, pp. 718–728, 2022.

[23] K. Choudhary and B. DeCost, "Atomistic line graph neural network for improved materials property predictions. npj computational materials, 7 (1): 185," 2021.

[24] C. Chen, W. Ye, Y. Zuo, C. Zheng, and S. P. Ong, "Graph networks as a universal machine learning framework for molecules and crystals," Chemistry of Materials, vol. 31, no. 9, pp. 3564–3572, 2019.

[25] F. Xie, T. Lu, S. Meng, and M. Liu, "Gptff: A high-accuracy out-of-the-box universal ai force field for arbitrary inorganic materials," Science Bulletin, 2024.

[26] A. H. Ashouri, W. Killian, J. Cavazos, G. Palermo, and C. Silvano, "A survey on compiler autotuning using machine learning," ACM Computing Surveys (CSUR), vol. 51, no. 5, pp. 1–42, 2018.

[27] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, and H. Shen, "and y. chen. 2018. tvm: An automated end-to-end optimizing compiler for deep learning," in 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), 2018, pp. 578–594.

[28] Z.-D. Zhang, M.-L. Tan, Z.-C. Lan, H.-C. Liu, L. Pei, and W.-X. Yu, "Cdnet: A real-time and robust crosswalk detection network on jetson nano based on yolov5," Neural Computing and Applications, vol. 34, no. 13, pp. 10 719–10 730, 2022.

[29] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen et al., "Ansor: Generating {High-Performance} tensor programs for deep learning," in 14th USENIX symposium on operating systems design and implementation (OSDI 20), 2020, pp. 863–879.

[30] D. Snider and R. Liang, "Operator fusion in xla: Analysis and evaluation," arXiv preprint arXiv:2301.13062, 2023.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., "Pytorch: An imperative style, high-performance deep learning library," Advances in neural information processing systems, vol. 32, 2019.

[32] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, "Degree-quant: Quantization-aware training for graph neural networks," arXiv preprint arXiv:2008.05000, 2020.

[33] Y. Wang, B. Feng, and Y. Ding, "Qgtc: accelerating quantized graph neural networks via gpu tensor core," in Proceedings of the 27th ACM SIGPLAN symposium on principles and practice of parallel programming, 2022, pp. 107–119.

[34] M. Ding, K. Kong, J. Li, C. Zhu, J. Dickerson, F. Huang, and T. Goldstein, "Vq-gnn: A universal framework to scale up graph neural networks using vector quantization," Advances in Neural Information Processing Systems, vol. 34, pp. 6733–6746, 2021.

[35] B. Feng, Y. Wang, X. Li, S. Yang, X. Peng, and Y. Ding, "Sgquant: Squeezing the last bit on graph neural networks with specialized quantization," in 2020 IEEE 32nd international conference on tools with artificial intelligence (ICTAI). IEEE, 2020, pp. 1044–1052.

[36] S. Hu, W. Zhang, Q. Sha, F. Pan, L.-W. Wang, W. Jia, G. Tan, and T. Zhao, "Rlekf: an optimizer for deep potential with ab initio accuracy," in Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence, ser. AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. [Online]. Available: https://doi.org/10.1609/aaai.v37i7.25957

[37] H. Si-Yu, Z. Yuan-Chang, Z. Tong, W. Lin-Wang, J. Wei-Le, and T. Guang-Ming, "Neural network force field training based on reorganized layer-wised extended kalman filter," Journal of Software, pp. 1–17.

[38] I. S. Novikov, K. Gubaev, E. V. Podryabinkin, and A. V. Shapeev, "The mlip package: moment tensor potentials with mpi and active learning," Machine Learning: Science and Technology, vol. 2, no. 2, p. 025002, dec 2020. [Online]. Available: https://dx.doi.org/10.1088/2632-2153/abc9fe

[39] A. Thompson, L. Swiler, C. Trott, S. Foiles, and G. Tucker, "Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials," Journal of Computational Physics, vol. 285, pp. 316–330, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999114008353

[40] S. Hu, T. Zhao, Q. Sha, E. Li, X. Meng, L. Liu, L.-W. Wang, G. Tan, and W. Jia, "Training one deepmd model in minutes: a step towards online learning," in Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, ser. PPoPP '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 257–269. [Online]. Available: https://doi.org/10.1145/3627535.3638505