Massively Parallelization Strategy for Material Simulation Using High-Dimensional Neural Network Potential

Cheng Shang,* Si-Da Huang, and Zhi-Pan Liu*

The potential energy surface (PES) calculation is the bottleneck for modern material simulation. The high-dimensional neural network (HDNN) technique emerged recently appears to be a problem solver for fast and accurate PES computation. The major cost of the HDNN lies at the computation of the structural descriptors that capture the geometrical environment of atoms. Here, we introduce a massive parallelization strategy optimized for our recently developed power-type structural descriptor. The method involves three-levels: from the top to the bottom the parallelization is over atoms first, then, over structural descriptors and finally over the *n*-body functions. We illustrate the parallelization method in a boron crystal system and show that the parallelization efficiency is maximally 100%, 58%, and 34% at each level. © 2018 Wiley Periodicals, Inc.

DOI:10.1002/jcc.25636

Introduction

Recent years have seen rapid development of high dimensional neural network (HDNN) technique originally proposed by Belher and Parrinello for constructing the potential energy surface (PES) of complex materials.[1,2] The HDNN has demonstrated its great potential for evaluating PES with a high accuracy (comparable to the first principles calculations) and low computational cost, for example, four orders of magnitude times faster as reported recently.[3,4] The computation costs of HDNN involve mainly two parts. (i) the numerical computation of complex functions, known as structural descriptors (SDs), that are used to describe the geometry and thus discriminate different structures; (ii) the evaluation of NN. As the part (i) is often the bottleneck for large-scale HDNN simulations, it is critical for developing new methods to reduce the cost of SD computation, in particular by improving the parallel efficiency.

The SD is the key element to establish the correlation between a structure and its energy. Through years, many different SDs have been developed, for example, extended-connectivity fingerprint,[5] Coulomb matrix,[6–9] graph convolution,[10,11] and symmetry function.[1,2] Paradoxically, with the increase of the ability of SD for describing complex PES, the function forms become more complicated and, thus, more computational demanding. What is even worse, the parallelization solution for speeding up SD remains to be a field not explored and no standard and efficient algorithm is available. Conversely, the evaluation of NN has been well studied in computer science and a number of highly efficient commercial packages for CPU and GPU, such as TensorFlow, Caffe, and MXNet[12,13] are available on market. These methods can utilize thousands of GPU cores to speed up the activation function computation in hundreds of neurons. Considering of these, to develop efficient parallelization algorithm for SD computation is an urgent task for the upcoming HDNN applications.

As one of the most sophisticated but also successful SD, Belher-type SDs[1,2] utilizes the Gaussian functions and trigomical function for describing the two-body and three-body interactions, which has a great similarity with the energy functions involving bond distance and bond angles utilized in traditional empirical potentials.[14] Our recent work[15] shows that Belher-type SDs may still not be enough for describing highly complex covalent bonded systems, such as Boron solid. Instead, we have developed a new series of SD, that is, power-type structural descriptors (PTSD). By utilizing the PTSD, the NN potential for boron global PES manages to reach the RMS accuracy of 12 meV/atom in energy.[15]

In this work, we focus on the parallelization scheme of PTSD. A three-level stradegy based on message passing interface (MPI) is designed to distribute the computational task into thousands of processors as even as possible. Depending on the number of atoms in system and the available processors, the parallelization scheme may choose to over atom or over PTSD. Only when the number of processors exceed the number of atoms, we split different PTSDs into different processors according to the estimated computational cost of each PTSD. The effciency of the current method is illustrated finally in boron solids examples.

A Brief Introduction of HDNN and PTSD

Before describing our parallelization strategy, we first briefly outline the HDNN architecture. Figure 1 shows that in HDNN the total energy of a system is a sum of independent atomic energy. The computation of each atomic energy requires the

C. Shang, Si-D. Huang, Zhi-P. Liu
Collaborative Innovation Center of Chemistry for Energy Material, Shanghai Key Laboratory of Molecular Catalysis and Innovative Materials, Key Laboratory of Computational Physical Science (Ministry of Education), Department of Chemistry, Fudan University, Shanghai 200433, China
E-mail: cshang@fudan.edu.cn or zpliu@fudan.edu.cn

Contract Grant sponsor: Shanghai Pujiang Program; Contract Grant number: 16PJ1401200; Contract Grant sponsor: Science and Technology Commission of Shanghai Municipality; Contract Grant number: 08DZ2270500; Contract Grant sponsor: National Science Foundation of China; Contract Grant numbers: 91645201, 21603035, 91745201, 21533001; Contract Grant sponsor: Science Challenge Project; Contract Grant number: TZ2018004; Contract Grant sponsor: National Key Research and Development Program of China; Contract Grant number: 2018YFA0208600

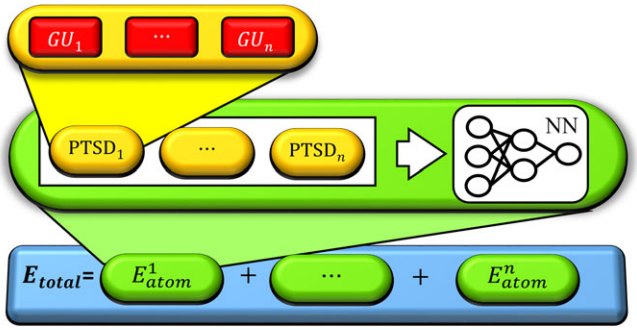

Figure 1. Scheme of the computation hierarchy in the PTSD-based HDNN. The total energy of a system (blue) is a sum over all atomic energy (green). The atomic energy involves descriptor computation (orange) and an NN evaluation (white). The descriptor computation includes contributions from each *n*-body function (group unit) computation (red). [Color figure can be viewed at wileyonlinelibrary.com]

calculation of the SDs of the associated atom followed by the standard feedforward-backpropagation NN evaluation. The SDs serve as the input layer of NN. For the PTSD, there are six types, namely $\mathbf{S^1}$ to $\mathbf{S^6}$ and they have the following mathematic forms (more details can be found in the Ref. [15].)

$$f_c(r_{ij}) = \begin{cases} 0.5 \times \tanh^3\left[1-\frac{r_{ij}}{r_c}\right], for\ r_{ij} \le r_c \\ 0 \quad for\ r_{ij} > r_c \end{cases} \quad (1)$$

$$R^n(r_{ij}) = r_{ij}^n \cdot f_c(r_{ij}), \quad (2)$$

$$\boldsymbol{S_i^1} = \sum \mathrm{GU}_1 = \sum_{j\neq i} R^n(r_{ij}), \quad (3)$$

$$\boldsymbol{S_i^2} = \left[\sum_{m=-L}^{L}\left|\sum \mathrm{GU}_2\right|^2\right]^{\frac{1}{2}} = \left[\sum_{m=-L}^{L}\left|\sum_{j\neq i} R^n(r_{ij})\,Y_{Lm}(r_{ij})\right|^2\right]^{\frac{1}{2}} \quad (4)$$

$$\boldsymbol{S_i^3} = 2^{1-\zeta}\sum \mathrm{GU}_3 = 2^{1-\zeta}\sum_{j,k\neq i}\left(1+\lambda\cos\theta_{ijk}\right)^{\zeta}\cdot R^n(r_{ij})\cdot R^m(r_{ik})\cdot R^p(r_{jk}), \quad (5)$$

$$\boldsymbol{S_i^4} = 2^{1-\zeta}\sum \mathrm{GU}_4 = 2^{1-\zeta}\sum_{j,k\neq i}\left(1+\lambda\cos\theta_{ijk}\right)^{\zeta}\cdot R^n(r_{ij})\cdot R^m(r_{ik}), \quad (6)$$

$$\boldsymbol{S_i^5} = \left[\sum_{m=-L}^{L}\left|\sum \mathrm{GU}_5\right|^2\right]^{\frac{1}{2}} = \left[\sum_{m=-L}^{L}\left|\sum_{j,k\neq i} R^n(r_{ij})\cdot R^m(r_{ik})\cdot R^p(r_{jk})\cdot\left(Y_{Lm}(\boldsymbol{r}_{ij})+Y_{Lm}(\boldsymbol{r}_{ik})\right)\right|^2\right]^{\frac{1}{2}}, \quad (7)$$

$$\boldsymbol{S_i^6} = 2^{1-\zeta}\sum \mathrm{GU}_6 = 2^{1-\zeta}\sum_{j,k,l\neq i}\left(1+\lambda\cos\delta_{ijkl}\right)^{\zeta}\cdot R^n(r_{ij})R^m(r_{ik})R^p(r_{il}), \quad (8)$$

Each PTSD can be considered as a sum of the *n*-body functions, named as the group unit (GU), also see Figure 1. The power function $R^n(r_{ij})$ represents the radial function in the GU. In the equations, r_{ij} is the distance of all atomic pairs, r_c is the cutoff, beyond which the value of eq. 1 is equal to zero, $Y_{Lm}(r_{ik})$ is the spherical harmonic function, and *n*, *m*, *p*, λ, and ζ are power parameters. Depending on the functional form of GU, the PTSD $\boldsymbol{S_i^1}$, $\boldsymbol{S_i^2}$, $\boldsymbol{S_i^3}$, $\boldsymbol{S_i^4}$, $\boldsymbol{S_i^5}$, and $\boldsymbol{S_i^6}$ can be considered as 2-body, 3-body, and 4-body functions, respectively, where the S_i^2 and S_i^5 involve also the spherical function.

Because the exact functional form differs, the computational cost of different type of PTSD in real systems is in fact very different. In Table 1, we show the typical cpu time for $\mathbf{S^1}$ to $\mathbf{S^6}$ PTSDs at three different cutoffs (r_c) from 2.0 to 6.0 in a boron crystal. With the increase of cutoffs, the number of group units (*ngu*) for PTSD may also increase (more neighbors are included). All cpu time is relative to a GU computation time of $\mathbf{S^1}$ ($T0^1$) (on a single CPU it takes around 0.0016 s.). As shown, the $T0^x$for six PTSD types $\mathbf{S^1}$–$\mathbf{S^6}$, follows the order $T0^1<T0^4<T0^3<T0^6<T0^2<T0^5$. Obviously, with the increase of *n*, the computation of *n*-body functions becomes generally more expensive; the $\mathbf{S^2}$ and $\mathbf{S^5}$ type PTSD with spherical functions have the largest *T*0.

The Td^i, the time to compute an *i*th PTSD with a type $\mathbf{S^x}$, can then be derived as the unit time of $\mathbf{S^x}$ ($T0^x$) multiplying the number of group units (*ngu*), that is, $Td^i = ngu \times T0^x$. For the boron crystal, *Td* at different cutoffs are also compared in Table 1. In general, those related to the 4-body type are the slowest, followed by the 3-body types and 2-body types. This is mainly due to the rapid increase of *ngu* from 2-body to 3-body and to 4-body PTSD. For the most expensive 4-body type $\mathbf{S^6}$, the *ngu* is typically an order of magnitude larger than that of the other types at the same cutoff. It is noted that although *T*0 of $\mathbf{S^2}$ type is high, the *ngu* of $\mathbf{S^2}$ type is small, making $\mathbf{S^2}$ a relatively cheap PTSD type despite the computation of spherical function.

Parallel Computation Strategy of PTSD-Based HDNN Potential

Considering that each atom has a similar set of PTSDs and different PTSDs may have very different computational cost (as measured by Td^i), we design a three-level parallelization scheme, which corresponds to the hierarchy architecture in HDNN computation shown in Figure 1, namely (i) A-para, atomic parallelization, (ii) D-para, PTSD parallelization, and (iii) G-para, GU parallelization. The purpose of the three-level parallelization scheme is to minimize the communication among processors and to load each processor as equally as possible that reduces the waiting time of processor. These two are essential for achieving a high parallel efficiency under the MPI framework.

Table 1. Typical computational cost of each type of PTSD

		$Td^{[b]}$		
Type	$T0^{[a]}$	r_c = 2.0	r_c = 4.0	r_c = 6.0
$\mathbf{S^1}$	1	12	78	248
$\mathbf{S^2}$	10	125	811	3×10^3
$\mathbf{S^3}$	2	152	7×10^3	7×10^4
$\mathbf{S^4}$	3	79	4×10^3	5×10^4
$\mathbf{S^5}$	17	406	2×10^4	2×10^5
$\mathbf{S^6}$	5	1×10^3	3×10^5	1×10^7

[a] The relative time to compute a GU involved in each PTSD. The *T*0 of $\mathbf{S^1}$ ($T0^1$) is set to be the unity.
[b] The typical time to compute each type of PTSD at different cutoff value.

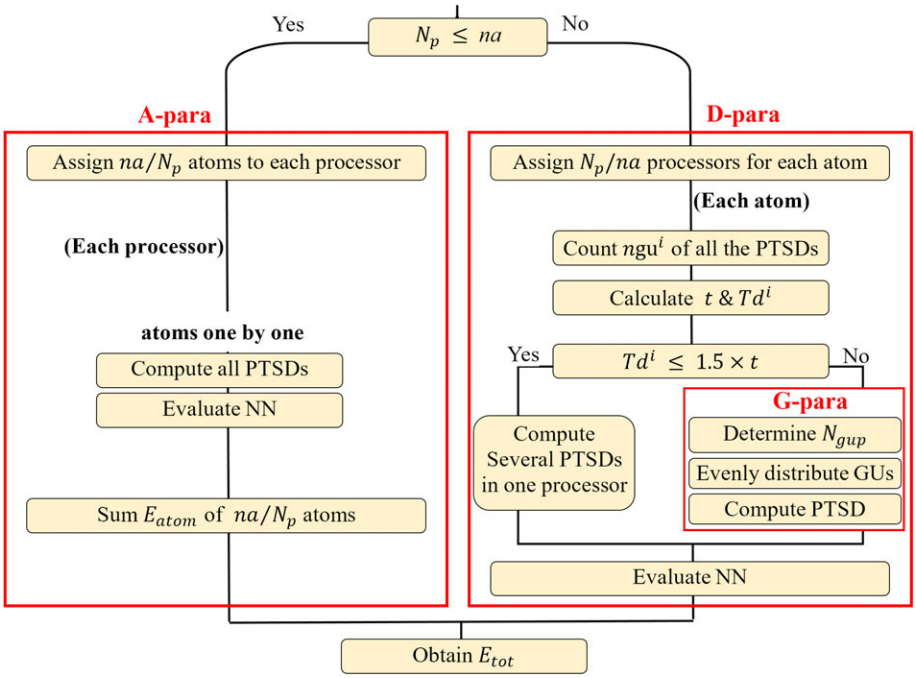

Figure 2. Flow chat of PTSD-based HDNN parallelization under the MPI framework. [Color figure can be viewed at wileyonlinelibrary.com]

In our implementation, the three-level parallelization can be selected automatically during the simulation. Because each atom has a similar set of PTSDs, the computational cost for each atom is similar and the A-para should be chosen preferentially. This can be done by comparing the number of atoms in system, *na*, with the number of processors, *Np*. As long as $N_p \leq na$, the A-para will be selected. This is shown schematically in the left-hand side of the flow chart in Figure 2, as explained in the following.

A-para scheme

The atom parallelization is natural to HDNN framework and thus technically simple in implementation. There are three basic steps.

- **A-1**. Distribute atoms to processors. Each processor needs to compute na/N_p atoms.
- **A-2**. Compute the atomic energy of each atom in one processor using HDNN. This involves the computation of PTSDs and NN evaluations in one processor.
- **A-3**. Sum up the atomic energies/forces computed in the same processor.

Conversely, if the number of processors N_p exceeds the number of atoms, each atom can be assigned to multiple processors and the D-para scheme will be invoked to compute each atom, which means that every processor computes only a fraction of PTSDs of an atom. The G-para is carried out only for the PTSDs of high computational cost and such a high cost PTSD will be distributed over a few processors, each computing a fraction of GUs of the PTSD, as illustrated by the right-hand side of the flow chart in Figure 2.

In the D-para and G-para, a number of $N_{subp} = N_p/na$ processors is first allocated to every atom. Then, the computation time of each PTSD (Td^i) is estimated according to Table 1, and the average execution time of one processor (t) can be derived as $t = \sum Td^i/N_{subp}$, being the sum of the estimated computational time Td^i for each PTSD divided by N_{subp}. In Figure 3a, we illustrate the computation time of each PTSD (Td^i) using rectangular box that varies in size from one to another (one box represents a PTSD), and the average time for processors using color bar that has the same length (one individual color bar represents one processor). One simple way of distributing the computational task would be keeping each processor loading as t, and splitting the GUs of PTSDs to multiple processors (as shown in Fig. 3a). Apparently, in the simple distribution, the loading of processor is even, but there is a significant cost in the data communication between processors to exchange GU values. To overcome this problem, we have designed three rules to assign PTSDs into processors to balance the processor loading and communication cost between processors. The algorithm is described as follows.

Starting from the first PTSD ($j = 1$) and the first processor ($n = 1$), the accumulated PTSD computing time Ta^j for the j PTSD and the accumulated processor execution time ta^n at the n processor [eqs. 9 and 10] are evaluated sequentially until the final PTSD is reached.

$$Ta^j = \sum_{i=1}^{j} Td^i, \tag{9}$$

$$ta^n = n \times t, \tag{10}$$

If $Ta^j < ta^n$, it indicates the n processors can finish j PTSDs. We continue to compute eq. 9 by incrementing $j = j + 1$. If $Ta^j > ta^n$, three rules are used to make decision how to distribute the jth PTSD, for example, whether the G-para strategy is used to compute the jth PTSD. These rules are illustrated in Figure 3 and explained in the following.

- **Rule 1**: If Ta^j exceeds the expected execution time of processor by less than the minimum value of $t/2$ and

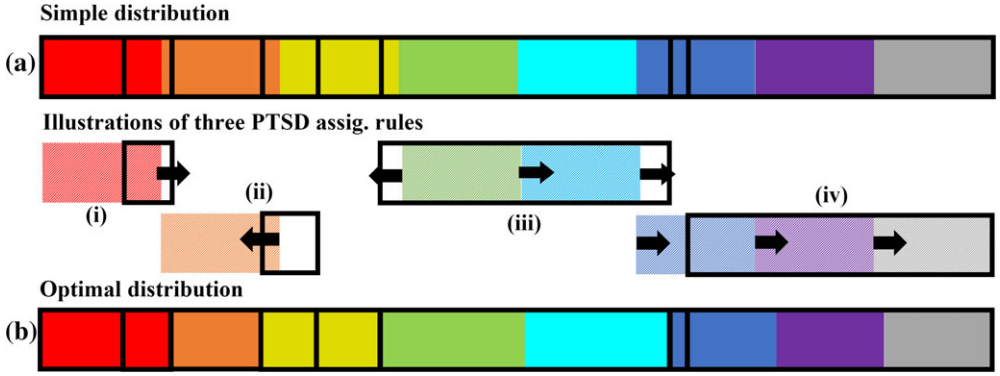

Figure 3. Schematic illustration of the PTSD parallel computation scheme. a) Simple distribution with equal loading (the same cpu time *t*) on each processor but the computation of PTSDs are split into multiple processors. Each color bar represents the loading of a processor. A black box represents the required cpu time for computing a PTSD. i–iv) are schematic illustration of three PTSD assignment rules, where the shaded bar represents the loading of the processor that needs to be modified. b) Optimal distribution with minimum communication among processors according to the three assignment rules. [Color figure can be viewed at wileyonlinelibrary.com]

$Td^j/2$, the *j*th PTSD is assigned totally to the current *n* processor. As the result of Rule 1, the actual execution time of the current processor (e.g., red bar in Fig. 3i) will be elongated. After the assignment, we let $j = j + 1$ and $n = n + 1$.

- **Rule 2:** If Ta^jexceeds the expected execution time of processor by more than $t/2$ or $Td^j/2$, the *j*th PTSD is assigned to the next processor (yellow bar, in Fig. 3ii) by increasing the processor count *n* ($n = n + 1$, the yellow bar becomes the current processor). As a result of Rule 2, the actual execution time of the previous processor (orange bar) is reduced. After the assignment, we let $j = j + 1$.
- **Rule 3:** If Ta^j exceeds the expected execution time of processor by more than $1.5 \times t$, suggesting it can be computed by multiple processors, the G-para strategy is invoked and the GUs of the *j*th PTSD are evenly distributed to the subsequent processors by increasing continuously the processor count *n* until Rule 3 breaks ($n = n + k$). The processor *n* is allocated to the *j*th PTSD only if its remaining expected execution time of processor ($Ta^j - ta^n$) is larger than $t/2$. For example, in Figure 3iii where the processor *n* is not allocated to *j*th PTSD, the box (*j*th PTSD) overlaps mainly with the green, cyan bars, and only slightly with the yellow/blue bar. As a result of Rule 3, the green and cyan bars are allocated to *j*th PTSDs. In Figure 3iv where the processor *n* is allocated to *j*th PTSD, the box (*j*th PTSD) overlaps largely with the last three bars. Hence, the blue, purple, and gray bars are allocated to *j*th PTSDs. After the assignment, we let $j = j + 1$.

Overall, the D-para and G-para schemes are summarized as follows.

D-para and G-para schemes

The SD parallelization is chosen if N_p is larger than *na*.

- **D-1**. Distribute processors to atoms. Each atom will be computed by $N_{subp} = N_p/na$ processors.
- **D-2**. Count the number of GUs (*n*gu) for all the PTSDs of the atom in parallel over N_{subp} processors.
- **D-3**. Estimate the average execution time (*t*) for each processor, $t = \sum Td^i/N_{subp}$.
- **D-4.** Assign PTSDs to processors according to three assignment rules (**Rule 1–3** above). If G-para is invoked, continue to G-1, otherwise to D-5.
- **G-1**. Assign $N_{gup} = Td^i/t$ processors to compute the PTSD, and each processor is assigned with ngu^i/N_{gup} GUs.
- **G-2**. Calculate the GU contributions to the PTSD on each processor.
- **G-3**. Collect GU contributions from each processor and evaluate the PTSD.
- **D-5**. Collect all PTSDs from N_{subp} processors and evaluate the atomic energy/forces using HDNN.

Benchmarks

Parallel efficiency

We use Boron crystals as an example to test our parallelization strategy (Fig. 4). For the testing purpose, the crystal used is a three-dimensional buckled honeycomb structure.[16] Each lattice contains four atoms with exactly the same chemical environment (Fig. 4a), thus, the same computational cost in HDNN evaluations. Our boron NN potential is taken from the previous work,[15] which were generated from the global optimization of various boron structures, ranging from bulk to layer and to clusters. The input layer of boron NN potential contains 173 PTSDs, which contain 39 $\mathbf{S^1}$, 36 $\mathbf{S^2}$, 16 $\mathbf{S^3}$, 52 $\mathbf{S^4}$, 18 $\mathbf{S^5}$, and 12 $\mathbf{S^6}$. To test the parallel efficiency at three different levels, boron crystals used have different sizes, that is, 4, 28, 140, and 560 atoms per cell. All NN calculations run on a same 560-core cluster (28 core/node × 20 nodes, Intel Xeon E5-2695 v3 CPU).

We also compared our NN calculations with first principles calculations using VASP[17,18] on the parallel efficiency. In VASP calculations, the kinetic energy cutoff was 450 eV, and the projector augmented wave (PAW) pseudopotential[19] was utilized to describe ionic core electron. The exchange-correlation

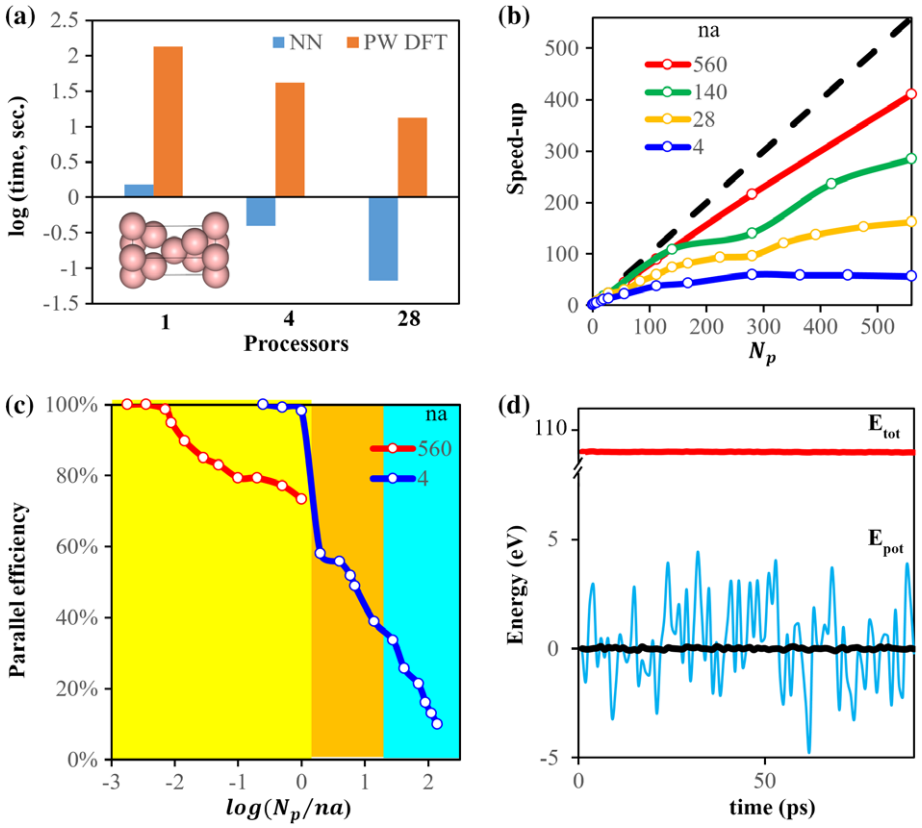

Figure 4. a) Comparison of the computation time for a 28-atom Boron crystal using NN and DFT with plane wave basis set (PW DFT). The *x* axis is the logarithm of time in the unit of seconds. The insert shows the structure of the boron crystal. b) Speed-up of HDNN computation of boron crystals. The *x* axis ($\boldsymbol{N_p}$) is the number of processors and the *y* axis is the speed-up. c) Parallel efficiency of HDNN computation of boron crystals. The yellow, orange, and cyan background represent three-level parallelization using A-para, D-para, and G-para, respectively. d) MD trajectory of a 1120-atom Boron crystal in the microcanonical (NVE) ensemble by highlighting the potential energy (E_{pot}) and the conserved total energy (E_{tot}). The stepsize is 1 fs. [Color figure can be viewed at wileyonlinelibrary.com]

functional for DFT was GGA-PBE functional. The first Brillion zone *k*-point sampling utilizes an automated Monkhorst–Pack scheme[20] with the mesh determined by 20 times of the reciprocal lattice vectors.

The parallel efficiency for NN calculations and DFT calculations are compared in Figure 4a. The benchmark system is a single-point energy calculation of a 28-atom crystal. For NN, the computing time is 1.5, 0.4, and 0.07 s on 1, 4, and 28 processors, respectively. For DFT, it is 135, 41, and 13 s, respectively. As a result, over 28 processors NN is 200 times faster than the DFT and the corresponding parallel efficiency are 81% for NN and 36% for DFT. The speed-up on 28 cores is basically the maximum speed-up for DFT computation of this system, however, as we will show in below, the NN can run even faster with more processors.

To examine our parallel strategy in detail, we have run a series of calculations on maximum 560 processors for crystals containing 4, 28, 140, and 560 atoms per cell. The results are shown in Figures 4b and 4c. For the 4-atom crystal, the maximum speed-up is 60, achieved over 280 processors (Fig. 4b), which indicates a high parallelization ability of NN calculations: 70 processors for each atomic energy computation. For systems containing 28, 140, and 560 atoms the maximum speed-up is 162, 285, 411, respectively, all achieved over 560 processors (Fig. 4b). It suggests that a higher speed-up is expected for these large systems if more processors are available.

We also note that the four curves in Figure 4b exhibit quite different slopes that also depend on the number of processors. This is apparently due to the different performance of three levels of parallelization and can be better understood from the analyses in Figure 4c. The A-para has the highest parallel efficiency, performed for all systems when $N_p \le na$ (yellow region in Fig. 4c), for example, 73% for 560-atoms over 560 processors and 98% for 4-atoms over 4 processors. This leads to a largely linear scaling for the 560-atom system over 560 processors (the red curve in Fig. 4b).

For 140-atom, 28-atom, and 4-atom systems, the D-para is also involved when $na < N_p < 17 \times na$ (illustrated as the orange region in Fig. 4c for 4-atom system). The D-para exhibits the medium parallel efficiency, for example, 58% over 8 processors and 39% over 56 processors for 4-atom system. The green curve in Figure 4b, corresponding to the 140-atom system, flats in between 140 and 280 processors due to the switch from A-para to D-para.

For 28-atom and 4-atom systems, the G-para is also involved when $N_p \ge 17 \times na$ (illustrated as the cyan region in Fig. 4c for 4-atom system). As expected, the parallel efficiency of G-para is the lowest in three levels, being 34% over 112 processors and 21% over 280 processors for the 4-atom system (Fig. 4c).

The above results show that the D-para has a much poorer parallel efficiency compared to the A-para. Compared to the A-para, the D-para has the additional communication cost in distributing and collecting the information between PTSDs among $\boldsymbol{N_{subp}}$ processors. Conversely, the switch-on of the G-para does not reduce obviously the parallel efficiency compared to that of the D-para. This is illustrated in Figure 4c, where the blue curve follows almost the same slope when crossing from the orange to cyan region. It indicates that the communication between the GUs computations as involved in the G-para for the most expensive PTSDs is in fact not the rate-determining step and the overall cost of parallelization is still dominated by the D-para.

Long time MD simulation

Finally, we have performed a molecular dynamics simulation of a 1120-atom crystals in the micro canonical (NVE) ensemble for 0.1 ns on 560 cores with verlet algorithm. The initial temperature is 1000 K and the timestep is 1 fs. The neighbor list is updated every timestep and, thus, the parallelization scheme is determined every timestep. The simulation trajectory is shown in Figure 4d. The total energy conserves well in the 0.1 ns simulation, suggesting the numerical stability of the current parallelization method. The simulation takes 6 h and 17 min in total. We estimate that it requires more than 1.2×10^4 h if the same job is performed using DFT.

Conclusions

In summary, we have introduced a parallelization strategy for HDNN computation with the newly developed PTSD as the input layer. The parallelization framework is designed to be hierarchical with three-levels, namely over atoms (A-para), over SDs (D-para), and over the n-body functions (G-para). Such an implementation can achieve linear scaling of the computational cost for large systems with thousands of atoms, where the communication cost between processors is maximally reduced.

Using Boron solids as examples, we show that this parallel strategy can gain a high speed up on at maximum 70 processors per atom. The highest speed-up is 411 for a 560-atom crystal using A-para; 284.5 for a 140-atom crystal using D-para; 161.7 for a 28-atom crystal and 60 for a 4 atom crystal using G-para.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (2018YFA0208600), the Science Challenge Project (TZ2018004), National Science Foundation of China (21533001, 91745201, 21603035, 91645201), the Science and Technology Commission of Shanghai Municipality (08DZ2270500), Shanghai Pujiang Program (16PJ1401200).

Keywords: neural network · parallelization · structure descriptor

How to cite this article: C. Shang, S.-D. Huang, Z.-P. Liu. *J. Comput. Chem.* **2019**, *40*, 1091–1096. DOI: 10.1002/jcc.25636

[1] J. Behler, M. Parrinello, Phys. Rev. Lett. **2007**, 98, 146401.
[2] Behler, J. J. Chem. Phys. **2011**, 134, 074106, , 074106.
[3] S.-D. Huang, C. Shang, X.-J. Zhang, Z.-P. Liu, Chem. Sci. **2017**, 8, 6327.
[4] S. Ma, S.-D. Huang, Y.-H. Fang, Z.-P. Liu, ACS Appl. Energy Mater. **2018**, 1, 22.
[5] D. Rogers, M. Hahn, J. Chem. Inf. Model. **2010**, 50, 742.
[6] K. Yao, J. E. Herr, Parkhill, J., J. Chem. Phys. **2017**, 146, 014106.
[7] M. Rupp, A. Tkatchenko, K.-R. Müller, O. A. von Lilienfeld, Phys. Rev. Lett. **2012**, 108, 058301.
[8] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, A. Tkatchenko, Nat. Commun. **2017**, 8, 13890.
[9] K. Yao, J. E. Herr, S. N. Brown, J. Parkhill, J. Phys. Chem. Lett. **2017**, 8, 2689.
[10] J. N. Wei, D. Duvenaud, A. Aspuru-Guzik, ACS Cent. Sci. **2016**, 2, 725.
[11] H. Altae-Tran, B. Ramsundar, A. S. Pappu, V. Pande, ACS Cent. Sci. **2017**, 3, 283.
[12] Martín Abadi, A. A., Paul Barham, Eugene Brevdo,; Zhifeng Chen, C. C., Greg S. Corrado, Andy Davis,; Jeffrey Dean, M. D., Sanjay Ghemawat, Ian Goodfellow,; Andrew Harp, G. I., Michael Isard, Rafal Jozefowicz, Yangqing Jia,; Lukasz Kaiser, M. K., Josh Levenberg, Dan Mané, Mike Schuster,; Rajat Monga, S. M., Derek Murray, Chris Olah, Jonathon Shlens,; Benoit Steiner, I. S., Kunal Talwar, Paul Tucker,; Vincent Vanhoucke, V. V., Fernanda Viégas,; Oriol Vinyals, P. W., Martin Wattenberg, Martin Wicke,; Yuan Yu, A. X. Z.TensorFlow: Large-scale Machine Learning on Heterogeneous Systems. Available at: https://www.tensorflow.org/ (accessed Sep. 25th, 2018).
[13] Jia, Y. E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional Architecture for Fast Feature Embedding. Available at: https://github.com/BVLC/caffe (accessed Sep. 25th, 2018).
[14] V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg, C. Simmerling, Proteins **2006**, 65, 712.
[15] Huang, S.-D.; Shang, C.; Kang, P.-L.; Liu, Z.-P. Chem. Sci. **2018**, DOI: 10.1039/C8SC03427C
[16] W. H. Han, Y. J. Oh, D.-H. Choe, S. Kim, I.-H. Lee, K. J. Chang, NPG Asia Mater. **2017**, 9, e400.
[17] G. Kresse, J. Hafner, Phys. Rev. B Condens. Matter Mater. Phys. **1993**, 47, 558.
[18] G. Kresse, J. Furthmuller, Comput. Mater. Sci. **1996**, 6, 15.
[19] G. Kresse, D. Joubert, Phys. Rev. B Condens. Matter Mater. Phys. **1999**, 59, 1758.
[20] H. J. Monkhorst, J. D. Pack, Phys. Rev. B: Condens. Matter. Mater. Phys. **1976**, 13, 5188.

Received: 8 July 2018
Revised: 28 August 2018
Accepted: 9 September 2018
Published online on 10 November 2018