EPSR++: An Open Source, Empirical Potential Structure Refinement Neutron Data Analysis Framework, Supporting Parallel, Across-computer Cluster Nodes, and GPU Hardware Acceleration

Changli Ma\textsuperscript{a,b}, He Cheng\textsuperscript{a,b,*}, Taisen Zuo\textsuperscript{a,b}, Guisheng Jiao\textsuperscript{a,b}, Zehua Han\textsuperscript{a,b,c}

\textsuperscript{a}Institute of High Energy Physics, Chinese Academy of Sciences (CAS) Beijing 100049, China
\textsuperscript{b}Dongguan Neutron Science Center, Dongguan 523803, China
\textsuperscript{c}University of Chinese Academy of Sciences, Beijing 100049, China

Abstract

Empirical potential structure refinement (EPSR) is a neutron scattering data analysis algorithm and a software package. It was developed by the British spallation neutron source (ISIS) Disordered Materials Group, and aims to construct the most-probable atomic structures of amorphous materials. With the extensive construction of supercomputer clusters and the widespread use of graphics processing unit (GPU) acceleration technology, it is now necessary to update the EPSR with distributed memory architecture and GPU acceleration in the effort to improve its calculation speed. In this study, an open source framework EPSR++ is introduced. It can be paralleled across nodes within a computer cluster and supports GPU acceleration. In addition, the framework is programmed in the object-oriented language C++. Therefore, users can define a special simulation box, atoms, molecules, and random motion patterns that are convenient for their analyses. The framework was successfully tested with H\textsubscript{2}O standard samples, and was shown to reconstruct the correct microstructure. Parallel performance tests have shown that GPU acceleration has significant effects.

Keywords: Neutron Diffraction, Neutron Scattering, Empirical Potential Structure Refinement, Reverse Monte Carlo, Graphics Processing Unit, Message Passing Interface, Open Multi-Processing, C++

*Corresponding author.
E-mail address: machangli@ihep.ac.cn (Changli Ma), chenghe@ihep.ac.cn (He Cheng)
1. Introduction

Neutron total scattering for disordered materials is a powerful tool used to study the most probable atomic structures in an amorphous system. Since the introduction of the first total scattering spectrometer (TSS), which was developed at HELIOS in the 1970s, numerous important scientific problems have been solved with the use of these types of instruments, including the elucidation of the structures of water at high- and low-densities, and the observation of the heterogeneities in mixed alcohol–aqueous solutions.

The success of neutron total scattering is based on deuterization techniques. The neutron scattering pattern \( S(Q) \) is a weighted summation of the Fourier transforms of all the pair correlation functions (PDF),

\[
S(Q) = \sum_{\alpha=1}^{\alpha} c_{\alpha} b_{\alpha}^2 + \sum_{\alpha=1, \beta \geq \alpha} (2 - \delta_{\alpha\beta}) c_{\alpha} c_{\beta} b_{\alpha} b_{\beta} \int_0^\infty r^2 g_{\alpha\beta}(r) \frac{\sin(Qr)}{Qr} dr
\]

where \( \sum_{\alpha} c_{\alpha} b_{\alpha}^2 \) is a flat background, and \( (2 - \delta_{\alpha\beta}) c_{\alpha} c_{\beta} b_{\alpha} b_{\beta} \) is the weighted factor of different partial neutron structural factors, \( c_{\alpha/\beta} \) and \( b_{\alpha/\beta} \) are the molar ratio and neutron scattering length of atoms (nuclide) with types \( \alpha/\beta \), respectively. Therefore, there are \( M(M+1)/2 \) different \( g_{\alpha\beta}(r) \) functions in Eq. (1), and we must solve all of them first before the most probable atomic structure is obtained. Fortunately, neutron scattering can employ deuteration techniques. Deuterated samples have almost the same atomic structure as their hydrogenate forms, and each deuterated sample can thus generate different scattering patterns.

Computer simulations must be involved to solve the matrix of Eq. (1)s, because the number of deuterated samples is still smaller than \( M(M+1)/2 \). A direct way to obtain the microstructure of samples is to apply the reverse Fourier transformation to \( S(Q) \), which is traditionally referred to as the PDF method [1, 2]. The results of the PDF method can be interpreted in terms of a pair distribution functions applied directly only in monatomic samples, such as in inert gas liquids. In addition, the PDF method can yield a statistical average coordination number for one atom type around another, but cannot yield the direct visualization of the atomic structure of the amorphous samples [3, 4, 5, 6]. Compared with the PDF method, EPSR [3, 4] is a Monte Carlo (MC) method for neutron data analyses to reconstruct the atomic structures of samples. Because EPSR can provide a reliable and visualized atomic microstructure, it has been extensively used in recent years.

With the extensive construction of supercomputer clusters and the widespread use of graphics processing unit (GPU) acceleration technology, the object-oriented
language C++ and the distributed memory architecture [7] API message passing interface (MPI) mpich2 [8] are used to develop an open source framework EPSR++ to implement the EPSR algorithm. In the framework, users can define easily a new simulation box, atoms, molecules, and movement models using the class multiple inheritance mechanism. With mpich2 API, the framework can be paralleled across nodes of a supercomputer cluster. In addition, GPU hardware acceleration is supported by the compute unified device architecture (CUDA) [9, 10, 11, 12, 13, 14], and this allows the program to take advantage of commonly used GPU computing servers.

2. EPSR Principle and EPSR++ Algorithmic Flow

2.1. Brief Description of EPSR Principle

EPSR is essentially a MC energy minimization simulation method, but the difference from Metropolis MC is based on the fact that the atomic potentials used in simulations are based on experimental rather than on theoretical data. In EPSR, the atomic potential used in the MC simulation is divided into two categories i.e., “reference potential (RP)” and “empirical potential (EMP)” [3]. RP is similar to that used in molecular dynamic (MD) simulations, so the potential form and parameters can be obtained from all the atomic MD force fields, such as the optimized potential for liquid simulations (OPLS) [15, 16], assisted model building with energy refinement (AMBER) [17], chemistry at HARvard molecular mechanics (CHARMM) [18], and others [3, 19]. By contrast, EMP has no fixed form and it is used to reflect the differences of neutron structural factors between experiments and MC simulations ($\Delta S(Q)$). To be exact, EMP is the reverse Fourier transform of $\Delta S(Q)$. In the present form of EPSR, EMP is expressed based on a list of Poisson distributions in real space and their corresponding Fourier transforms in Q space. In short, RP is used to assign molecules with reasonable shapes, while EMP is used as a feedback parameter to lead the MC simulation in a consistently progressive manner with diffraction experimental data. In the EPSR simulations, only RP is used in the MC simulation at the beginning, and the potential changes of the system ($\Delta U$) are used as the selection criteria of molecules or atomic random movements. When the simulation reaches equilibrium, EMP is introduced to fit $\Delta S(Q)$ and is added to RP to continue the simulation. When the MC simulation with updated potential reaches equilibrium again, EPSR calculates EMP and updates the simulation potential once more. This process is repeated until the EMP becomes equal to zero, i.e., $\Delta S(Q)$ becomes very small.
2.2. EPSR++ Algorithmic Flow

The algorithmic flow of EPSR++ is shown in Fig. 1. Details of some processes are described below:

I. Define the simulation box, molecules, and atoms, select a parallel method, and recompile the program. Three basic C++ classes are defined in the framework to allow users to define special classes or objects:

- SimBox: It is used for generating a simulation box. It can be initialized with the molar mass of molecules, molecular numbers, density, etc. Users can define a special initial function of the simulation model box.

- Molecule: It is used to define molecules, intramolecular potentials and their movements, such as translation, rotation, etc. Users can define arbitrary molecular classes and can try special inter- and intramolecular movements for their analyses. This class makes the program very flexible and user-friendly.

- Atom: It is used to define an atom type in a simulation. Some basic atomic properties, such as the atom name, element name, isotope name, coordinate position, neutron scattering length, are defined. The coordinate position is defined using the Hep3Vector class of the class library for high energy physics (CLHEP) [20, 21] because the Hep3Vector has abundant functions to perform transition, rotation, distance, and angle calculations. Users need to initialize atoms in molecular objects rather than define new atom classes.

In addition, new atom potential and potential functions can be added by editing the C++ head and source files NDA_parameter_pot.h and NDA_func_pot.h.

After the classes for the special simulations have been defined, users need to edit the Makefile to select which parallel method will be used to accelerate the program *i.e.*, the use of the “-D USE_MPI,” “-D USE_OMP,” and “-D USE_GPU” compiler options indicate the employment of mpich2 API to make the program parallel among the different nodes of a computer cluster, while the use of OpenMP API makes the program parallel among many cores of a CPU and enables GPU hardware acceleration.

II. Edit Input File, Run Program: a text file needs to be input to the program to initialize the simulation box and run parameters, such as the molar mass of molecules, molecular numbers, density, maximal movement step, MC equilibrium criteria, EPMC finish criteria, etc.
Define SimBox, Molecule Class; Recompile Program

Edit Input File, Run Program

Initialize SimBox and Potential, and Define Variables

Is Use MPI?

Yes

Broadcast Box, Variables to PC Cluster

No

Parallel Method Select

OpenMP

MPI

GPU

MC

Reach Equilibrium?

No

Calculate \( g(r) \), \( S(Q) \), and Empirical Potential

Yes

Empirical Potential \( \approx 0 \) ?

Yes

EPMC

No

Reach Equilibrium?

No

Accumulate \( g(r) \) and \( S(Q) \)

Yes

Terminate Program

Fig. 1: Schematic of arithmetic flow of EPSR++. 
III. Initialize SimBox, Potential, and Define Variables: The program initializes the simulation box, reference potential arrays of all-atom type pairs with inputs parameters. The reference potential parameters are defined in a C++ head file, and users can edit or define these parameters conveniently. Potential functions can also be defined by users, including the L–J potential, electric field, and the potential truncation function.

If the program needs to be paralleled among different nodes of a computer cluster, the initialized simulation box and variables will be broadcast to different nodes with the \texttt{MPI\_Bcast()} function of mpich2.

IV. MC: The program performs MC simulations with reference potentials until equilibrium is reached. The program moves molecules or atoms in sequence or randomly. The potential energy variation of the simulation box ($\Delta U = U_{\text{after}} - U_{\text{before}}$) is used as the movement acceptance criterion. If $\Delta U < 0$, the movement is accepted. If $\Delta U > 0$, the movement is accepted with a probability $e^{-\Delta U/kT}$.

The computation consumes $g(r)$, is in direct proportion to the second order of the atom amount, and needs to be recalculated after every MC simulation step. Thus, this step represents the step with the highest consumption of the program’s calculation capacity. We designed three selectable parallel methods, \textit{i.e.}, OpenMP, MPI, and GPU, to accelerate the calculation. Users can select a suitable method according to their hardware and speed requirements.

V. Calculate $g(r)$, $S(Q)$, and EMP: When MC reaches equilibrium, the program can calculate the difference of the neutron structural factor $\Delta S(Q)$ between the simulation $S_{\text{sim}}(Q)$ and experiment $S_{\text{exp}}(Q)$. EMP is calculated by applying the Fourier transform to $\Delta S(Q)$. The program adds the EMP and RP together as the updated potential to perform the MC simulation.

The every visit preference Monte Carlo (EPMC) algorithm is very similar to MC with the exception that when the simulation reaches equilibrium, the program calculates the EMP again, adds it to the previous potential, and performs the simulation with the updated potential. When the EMP attains a very low value, the program stops to update the simulation potential.

VI. Accumulate $g(r)$ and $S(Q)$: When the execution stops, the simulation potential is updated. The program will still perform EPMC to accumulate simulation data to improve statistics until smooth $g(r)$ and $S(Q)$ curves are achieved.
obtained. With the exception of \( g(r) \) and \( S(Q) \), the program outputs a co-coordinate file which includes all the atoms in a text format, as used in the GROningen machine for chemical simulations (GROMACS) (with the suffix of \( .\text{gro} \)) [22], or as used in the large-scale atomic/molecular massively parallel simulator (LAMMPS) (with the suffix of \( .\text{xyz} \)) [23]. Accordingly, this file can be input to GROMACS or LAMMPS to calculate enthalpy, entropy, etc., and also can be visualized with visual molecular dynamics (VMD) [24, 25]. Users can define, calculate, and output any interesting variables which are related directly with the atomic structure of the sample by editing the source file.

3. Performance Test

Full hydrogen, full deuterated, and half deuterated H\(_2\)O samples are used to test the performance of the program regarding the correctness and computational speed.

3.1. Correctness

Based on the extended simple point charge model (SPC/E) [26] and the OPLS [15, 16] all atomic force fields, a control sample containing approximately 200 thousand atoms (125 Å) is generated with GROMACS for testing the correctness of the program. The comparison of PDF distributions and neutron diffraction spectra between the MD sample and the EPSR simulation are shown in Fig. 2.

As shown, the EPSR simulation results are consistent with the ground truth pertaining to the MD data of the control sample. Experimental neutron data of full hydrogen water, full deuterated water, and half deuterated water distributed with EPSR and GudRun [27] in the ISIS website are also used to test the program, and the simulation results are very similar with that in the right of Fig. 2. These show that the updated EPSR++ program can reconstruct the atomic structures of experimental samples correctly based on the neutron diffraction spectra.

3.2. Computational Speed

In the program, three different parallel methods (i.e., MPI, OpenMP, and GPU) are provided to increase the calculation velocity. A small computer cluster is used to test the speed performance of different methods. The cluster uses CentOS 7.3 as the operating system and has two nodes. Each node has two Intel Xeon Scalable Gold 6126 CPU (two Skylake–SP architectures, 12 cores, 24 threads 2.6 GHz, Turbo 3.7 GHz and a 19.25 MB L3 Intel smart cache), two Nvidia Tesla
Fig. 2: Left: PDF comparison between EPSR++ and MD control sample. The curves from top to bottom are the $g(r)$ plots of the O–H, H–H, O–O, respectively. Right: neutron diffraction spectrum comparison between EPSR++ and MD control samples. The curves from top to bottom are the $S(Q)$ plots of the full-deuterated, half-deuterated, and full-hydrogen samples. The solid lines are from the control samples, while the dashed lines are from the EPSR++ simulation.

V100 calculation GPU card and 128 GB double data rate (DDR4) error correcting code (ECC) registered shared memory. The two nodes are connected with an InfiniBand (IB) connector (data transmission speed can increase up to 56 Gb/s).

For the implementation of across nodes in a parallel configuration, mpich2 is used in the program. Mpich2 is based on the MPI standard and supports point-to-point and collective data communication among different nodes. Thus, it is very suited and is used highly efficiently in this program. For implementing a shared memory, multithread, parallel configuration within a computer or server node, Open Multi-Processing (OpenMP) is used in the C++ program. The OpenMP syntax supports the setting of a thread number dynamically, and a thread number cannot be known in advance in most cases. Thus, it is very convenient in programming. For supporting GPU hardware acceleration, a CUDA toolkit is used in the program.

A speed comparison of these acceleration methods is shown in Fig. 3, and a more detailed quantitative comparison is listed in Tab. 1. As shown in the figure and table, mpich2 and OpenMP can improve the speed based on the ratio of nodes or thread numbers, while the GPU can provide an excellent acceleration ratio. Most importantly, with the GPU acceleration, the program can simulate a system comprising > 1 million atoms. This is an essential improvement because it allows the program to simulate systems larger than 220 Å, so that it can analyze samples.
Table 1: Simulation speeds with different parallel methods and serial algorithms [steps/s].

| Atomic number | $3 \times 10^3$ | $3 \times 10^4$ | $1 \times 10^5$ | $2.5 \times 10^5$ | $1.2 \times 10^6$ | $3 \times 10^6$ | $1 \times 10^7$ | $3 \times 10^7$ |
|---------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| GPU           | 1562            | 9507            | 12412           | 10189           | 3917            | 1963            | 757.0           | 340.3           |
| MPI           | 2000            | 334.9           | 110.5           | 42.57           | —               | —               | —               | —               |
| OMP           | 1989            | 327.5           | 109.4           | 42.44           | —               | —               | —               | —               |
| CPU           | 1754            | 189.8           | 62.67           | 24.30           | —               | —               | —               | —               |

with polymers in all atomic models.

Fig. 3: Left: Calculation speed comparison among serial algorithm and different acceleration algorithms. Right: Calculation speed with GPU acceleration. The X coordinate denotes the atomic number of the simulation box, while the Y coordinate denotes the simulation steps in units of seconds.

4. Conclusions

The updated neutron diffraction data analysis software EPSR is reprogrammed with the object-oriented language C++. This made the program very flexible and friendly for users who need to define special molecules and MC random movement patterns. Potential functions and the corresponding parameters of the atomic force field can be modified or added by editing the C++ head and source files. In addition, C++ is an easy-to-read, high-level computer language, so users can try new algorithms and program flows to improve their analyses, and to calculate and output any variables related with the sample microstructures considered important for their analyses.
With the exception of parallel nodes within the server with OpenMP, parallel cross-different nodes of a computer cluster, and GPU hardware acceleration, were respectively supported with mpich2 and CUDA. Specifically, with GPU acceleration, the calculation speed was improved considerably and the program attained the capacity to analyze samples with macromolecular samples or nanoparticles of all atomic models.

Although the updated program is flexible for users and has a powerful calculation capacity, it was tested with a very limited number of control samples. Accordingly, in its current form, it is not a fully functional software package. The authors aspire to release it as an open-source toolkit framework for public use by interested scientists. In this sense, users will be able to contribute numerous new molecular classes, algorithms, and analyses routines in the future to make the program more powerful.

5. Acknowledgements

This work was supported by the National Key Research and Development Program of China [grant number 2017YFA-0403703]; and the National Natural Science Foundation of China (NSFC) [grant number U1830205, 21674020]. The authors thank the staff of the ISIS Disordered Materials Group for the explanations of the principles of EPSR and for providing experimental neutron diffraction data for testing of the program.

References

[1] J. Xu, L. Mei, W. Yin, X. Wang, W. Cai, Z. Li, T. Bo, H. Chen, B. Wang, Y. Chen, Nuclear Inst. and Methods in Physics Research, A 927 (2019) 161–168. doi:https://doi.org/10.1016/j.nima.2019.02.011.

[2] K. I. Shefer, Mater. Today 4 (2017) 11360–11363. doi:https://doi.org/10.1016/j.matpr.2017.09.009.

[3] R. L. Mcgreevy, L. Pusztai, Mol. Simulat. 1 (1988) 359–367. doi:https://doi.org/10.1080/08927028808080958.

[4] A. K. Soper, Chem. Phys. 202 (1996) 295–306. doi:https://doi.org/10.1016/0301-0104(95)00357-6.
[5] Empirical potential structure refinement (a user’s guide) version 25: February 2017, https://www.isis.stfc.ac.uk/OtherFiles/Disordered%20Materials/EPSR25%20Manual%202017-10.pdf, 2019 (Accessed 31 May 2019).

[6] A. K. Soper, K.J.Edler, BBA-Gen. Subjects 1861 (2017) 1652–1660. doi:https://doi.org/10.1016/j.bbagen.2017.02.028.

[7] E.-R. Hesham, A.-E.-B. Mostafa, Advanced Computer Architecture and Parallel Processing, John Wiley & Sons, Inc., Hoboken, New Jersey. Published simultaneously in Canada., 2005.

[8] Open mpi: Open source high performance computing, https://www.open-mpi.org/, 2019 (Accessed 31 May 2019).

[9] Z. Liping, G. Zhiqiang, C. Ruiwen, F. Yue, Z. Gaofeng, X. Benzhu, Comput. Graph. 80 (2019) 29–36.

[10] Matam Kiran Kumar, Mohammad RajabAbdel-Majeed, Murali Annavaram, Integration 66 (2019) 35–43. doi:https://doi.org/10.1016/j.vlsi.2018.12.006.

[11] J. R. Cheng, M. Gen, Comput. ind. Eng. 128 (2019) 514–525. doi:https://doi.org/10.1016/j.cie.2018.12.067.

[12] M. G. Awan, T. Eslami, F. Saeed, Comput. Biol. Med. 101 (2018) 163–173. doi:https://doi.org/10.1016/j.compbiomed.2018.08.015.

[13] Cuda c best practices guide, https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html, 2019 (Accessed 31 May 2019).

[14] Cuda c programming guide, https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, 2019 (Accessed 31 May 2019).

[15] W. L. Jorgensen, J. Tirado-Rives, J. Am. Chem. Soc. 110 (1988) 1657–1666. doi:https://doi.org/10.1021/ja00214a001.

[16] W. L. Jorgensen, D. S. Maxwell, J. Tirado-Rives, J. Am. Chem. Soc. 118 (1996) 11225–11236. doi:https://doi.org/10.1021/ja9621760.
[17] C. J. Dickson, B. D. Madej, Å. a. Skjevik, R. M. Betz, K. Teigen, I. R. Gould, R. C. Walker, J. Chem. Theory Comput. 10 (2014) 865–879. doi:https://doi.org/10.1021/ct4010307.

[18] B. B.R. Brooks, C.L. Brooks, A.D. MacKerell Jr, L. Nilsson, R.J. Petrella, A. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Cafli sch, L. Caves, Q. Cui, V. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, R. M. V. Ovchinnikov, E. Paci, R.W. Pastor, C.B. Post, J.Z. Pu, M. Schaefer, B. Tidor, H. L. Woodcock, X. Wu, W. Yang, D.M. York, M. Karplus, J. Comput. Chem. 30 (2009) 1545–1614. doi:https://doi.org/10.1002/jcc.21287.

[19] M. G. Martin, Fluid Phase Equilib. 248 (2006) 50–55. doi:https://doi.org/10.1016/j.fluid.2006.07.014.

[20] Clhep - a class library for high energy physics, http://proj-clhep.web.cern.ch/proj-clhep/, 2019 (Accessed 31 May 2019).

[21] L. Lönnblad, Comput. Phys. Commun. 84 (1994) 307–316. doi:https://doi.org/10.1016/0010-4655(94)90217-8.

[22] Gromacs, http://www.gromacs.org/, 2019 (Accessed 31 May 2019).

[23] Lammmps molecular dynamics simulator, http://lammps.sandia.gov, 2019 (Accessed 31 May 2019).

[24] Vmd–visual molecular dynamics, http://www.ks.uiuc.edu/Research/vmd/, 2019 (Accessed 31 May 2019).

[25] W. Humphrey, A. Dalke, K. Schulten, J. Mol. Graph. Model. 14 (1996) 33–38. doi:https://doi.org/10.1016/0263-7855(96)00018-5.

[26] P. Mark, L. Nilsson, J. Phys. Chem. A 105 (2001) 9954–9960. doi:https://doi.org/10.1021/jp003020w.

[27] Gudrun - routines for reducing total scattering data, https://www.isis.stfc.ac.uk/Pages/Gudrun.aspx, 2019 (Accessed 31 May 2019).