A Parallel Machine Learning Workflow for Neutron Scattering Data Analysis

Tianle Wang[a], Sudip K. Seal[b], Ramakrishnan Kannan[b], Cristina Garcia-Cardona[c], Thomas Proffen[d] and Shantenu Jha[a]
[a]Computational Science Initiative, Brookhaven National Laboratory, USA
[b]Computer Science and Mathematics Division, Oak Ridge National Laboratory, USA
[c]Computer, Computational and Statistical Sciences Division, Los Alamos National Laboratory, USA
[d]Spallation Neutron Source, Oak Ridge National Laboratory, USA

***Abstract*—As part of a larger effort, this work in progress reports the possible advantages of modifying conventional workflows used to generate labelled training samples and train machine learning (ML) models on them. We compare results from three different workflows using neutron scattering data analysis as the motivating application and report about 20% improvement in speedup, with no appreciable loss of model accuracy, over a baseline workflow.**

I. Introduction

Crystalline materials belong to seven crystallographic classes. A set of unit cell lengths, denoted by the parameter set $\{a, b, c\}$, and unit cell angles, denoted by the parameter set $\{\alpha, \beta, \gamma\}$ together define each crystallographic class. The parameters $\{a, b, c\}$ and $\{\alpha, \beta, \gamma\}$ are constrained to satisfy unique relations depending on the symmetry class the crystal belongs to. In this study, we only consider three classes, namely, tetragonal, trigonal and cubic symmetries. These symmetries are defined by the following parameter relations: tetragonal ($a = b \neq c$, $\alpha = \beta = \gamma = 90°$), trigonal ($a = b = c$, $\alpha = \beta = \gamma \neq 90°$) and cubic ($a = b = c$, $\alpha = \beta = \gamma = 90°$). A central goal of neutron scattering analysis is to determine the values of the parameters $\{a, b, c\}$ and $\{\alpha, \beta, \gamma\}$ from the pattern of neutrons scattered by the crystal and collected by detectors in the form of two-dimensional patterns called *Bragg profiles*.

It was recently shown [1] that trained ML models have the potential to predict these structural parameters directly from their Bragg profiles with acceptable accuracy for certain classes of materials. Training these ML models typically require large volumes of labelled training data. However, the availability of experimentally labelled neutron scattering data is very limited. To overcome this challenge, a physics-based simulator called GSAS-II [2] is used in this work to computationally simulate training samples of Bragg profiles labelled by $\{a, b, c\}$ and $\{\alpha, \beta, \gamma\}$. To do this, domain knowledge is used to limit the range of each of the six parameters and labelled samples are generated at known values of the six parameters within these ranges by stepping through the parameter space in pre-defined step sizes. Depending on the crystallographic symmetry being simulated, the total time to generate the training data can vary from a few hours to days. The task of generating the training data is shown by the blue circle in Fig. 1(a). The subsequent task of training ML models using the labelled samples to predict the parameters is shown by the red circle in Fig. 1(a). This sequence of tasks in which the data is generated via simulations followed by the training task is referred to as the *baseline* workflow here.

In this paper, we report the results of executing the preceding simulation and training tasks using two alternative workflows and compare their performance in terms of time-to-solution and training accuracy with those from the baseline workflow.

II. Workflow Middleware

RADICAL-Cybertools (RCT) is a set of software systems that serve as middleware for scientific computing. Specifically, RCT enables concurrent and sequential execution of applications with heterogeneous tasks on high-performance computing resources [3]. RCT has three main components: RADICAL-SAGA (RS), RADICAL-Pilot (RP) and RADICAL-Ensemble Toolkit (EnTK). EnTK supports the concurrent or sequential execution of tasks in an arbitrary priority relation (i.e., ensemble or pipelines of tasks). EnTK is implemented based on RP, which allows users to submit jobs to computing infrastructures and then use the resources acquired to execute one or more tasks. RP is implemented based on RS, a high-level interface to distributed infrastructure components like job schedulers, file transfer, and resource provisioning services. In this work, EnTK is used to

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan) A portion of this research used resources at the Spallation Neutron Source, a DOE Office of Science User Facility operated by the Oak Ridge National Laboratory. A portion of this research used resources at the Argonne Leadership Computing Facility, a DOE Office of Science User Facility operated by the Argonne National Laboratory. A portion of this research used resources at the Brookhaven National Laboratory, a DOE Office of Science National Laboratory. This research was sponsored by ExaLearn, an Exascale Computing Project, DOE.

build three workflows (baseline and two alternate workflows) and execute them on the Theta supercomputer housed in the Argonne Leadership Computing Facility. Each Theta node is an Intel KNL 7230 chipset with 64 cores and 192 GiB of DDR4 SDRAM and connected together by an Aries Dragonfly high-speed interconnect.

III. Workflow Description

The three workflows tested in this paper will be referred to as the baseline workflow ($\mathcal{W}_b$), the serial workflow ($\mathcal{W}_s$) and the parallel workflow ($\mathcal{W}_p$). These are shown in Fig. 1.

Baseline Workflow, $\mathcal{W}_b$: As mentioned earlier, the baseline workflow, shown in Fig. 1(a), consists of two tasks, the simulation task (blue circle), and the training task (red circle). Given a set of P resources, the baseline workflow is defined by the sequence of the following two tasks. In task 1, the P resources are used to simulate all N labelled samples of the training data set. This task is followed by task 2 in which the ML model $\mathcal{M}_b$ is trained to a desired accuracy, α, for a fixed number of epochs, say n_e. The N training samples are partitioned into $n_b = N/b$ batches where b is the batch size. The batch size used in this study is $b = 512$.

Serial Workflow, $\mathcal{W}_s$: The serial workflow, $\mathcal{W}_s$, is similar to the baseline workflow, $\mathcal{W}_b$. The difference is that the available P resources are used to simulate a smaller number of labelled samples at a time, then using all P resources to train the ML model $\mathcal{M}_s$ on this smaller training data set. This sequence of simulation and training (called a *phase* here) is repeated until all N training samples have been generated and used for training $\mathcal{M}_s$. This is shown in Fig. 1(b). The motivation for testing this workflow is two-fold: (a) since each simulation task only needs to generate a small portion of the entire training data, the number of phases can be tuned so that the data can fit into memory of the available P resources, and (b) the workflow allows introduction of active learning policies between the simulation and training tasks for intelligent sampling of the input parameter so that the training sample batches simulated in the $(k+1)^{th}$ phase can be based on the training result in the k^{th} phase.

Parallel Workflow, $\mathcal{W}_p$: The parallel workflow, $\mathcal{W}_p$, is a pipelined parallel version of the serial workflow. Here, we take advantage of the fact that the training task in any phase can be overlapped with simulation task of the next phase. See Fig. 1(c). Note that the overlap itself will not introduce significant benefit for the computational performance. This workflow mimics the case when the available resources consist of both CPU and GPU resources, as is the case in many hardware accelerated platforms. Typically, training tasks are offloaded to GPUs because most training algorithms ultimately translate to SIMD (single-instruction multiple-data) operations which are very efficiently executed on GPUs. $\mathcal{W}_p$ emulates a workflow that overlaps simulation tasks on CPUs and training tasks on GPUs. RCT is currently under development for GPU support. As such, $\mathcal{W}_p$ provides a mechanism to mimic a GPU-supported RCT workflow that overlaps CPU and GPU computations for simulation and training tasks, respectively, but *only* using CPU executions. The model trained using $\mathcal{W}_p$ is denoted by $\mathcal{M}_p$.

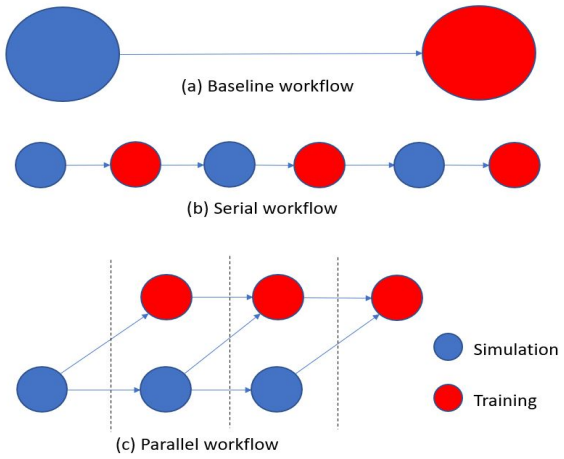


Figure 1: The baseline and alternate workflows.

IV. Model

The model $\mathcal{M}$ used in the training task in each of the workflows is a multitask network shown in Fig. 2. It uses the output from the first fully connected layer of the deep neural network classifier described in [1] to train both a regressor (using softmax) in addition to the original classification task. The classifier updates the weights using the error obtained from cross entropy loss while the regressor uses MSE loss for every batch.

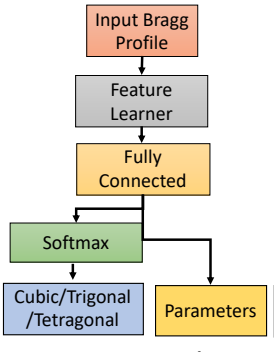


Figure 2: Model architecture.

V. Data Generation and Experiment Setups

In this preliminary work, we use a perovskite material called *barium titanate* ($BaTiO_3$), since it naturally exists only in three possible symmetry groups, namely, tetragonal, trigonal and cubic. Guided by domain knowledge, specific ranges of the parameters were chosen to generate the labelled samples for each symmetry class. Three experimental setups, referred to as *E1*, *E2* and *E3*, were used. These are described next.

For *E1* and *E2*, the range of a for the cubic symmetry (recall that for cubic symmetry, $a = b = c$) was chosen to be $[2.5, 5.5)$ with a step size of 0.005; for the trigonal symmetry, the parameter range for a was $[2.5, 5.5)$ with a step size of 0.05 and for α was $[20, 88) \cup [92, 120)$ with a step size of 0.5. For the tetragonal symmetry, the range for a was $[3.5, 4.5)$ with a step size of 0.005, and for c

was $[3.5, 4.5)$ with a step size of 0.01. For *E3*, the samples were simulated from the same parameter ranges as those of *E1* and *E2* with the following difference: the step size of a was 0.001 for cubic symmetry, the step size of a was 0.01 for trigonal symmetry and the step size of c was 0.002 for tetragonal symmetry. These choices ensured that the amount of data generated in *E3* was five times larger than that in *E1* and *E2*. We used a larger data set for *E3* because it used four times the resources than *E1* and *E2*.

Since the workflows $\mathcal{W}_s$ and $\mathcal{W}_p$ need to simulate the samples in phases, care is needed in how the samples are generated to ensure that all three workflows train the models using an identical distribution of training samples. In other words, if there are n phases in a workflow, then the training data samples need to be simulated in each of n partitions in such a manner that models trained in each phase sample training data from the same distribution as the baseline workflow to avoid any bias. A naive partitioning strategy in which the range of a parameter, say a, for the baseline simulation is $[a_{\min}, a_{\max})$ with a step size d, the range can be partitioned such that the k^{th} sub-range is $[a_{\min} + (k-1)\frac{\delta a}{n}, a_{\min} + k\frac{\delta a}{n})$ with the step size d, where $\delta a = a_{\max} - a_{\min}$. This naive partition introduces a bias in the training since the different phases sample the training data with different distributions. Here, we use the following bias-free policy for phased training sample generation: given the baseline range δa and step size d, the simulator simulates the samples $[a_{\min} + (k-1)d, a_{\max} - (n-k)d)$ with step size $n \cdot d$. This bias-free policy ensures that the training samples simulated in each phase do not overlap and the bias it introduces due to the difference in the distribution of different phases has an order of $O(d)$, which is negligible in this work since $d \ll \delta a$. This strategy is used for all the different input parameters for the simulation tasks.

In both $\mathcal{W}_s$ and $\mathcal{W}_p$, we choose the number of phases to be two in *E1* and *E2*, and four for *E3* in this preliminary study. Additionally, for all the symmetries, we apply the bias-free policy explained above to partition the input parameters and divide the baseline parameter ranges into two (in *E1* and *E2*) and four sub-ranges (in *E3*).

VI. Performance Results

The performance of the three workflows were compared across three different experimental setups, *E1, E2* and *E3*, as described before. Table I lists the setup configurations of each experiment. In each setup, the process ranks for the simulation tasks were mapped to cores of a Theta node and the process ranks for the training tasks were mapped directly to the Theta nodes, each with 64 CPU cores.

The primary goals of this preliminary work are to investigate the performance of the three workflows in the context of two main considerations. First, we wish to study how the model accuracy reacts to the different workflows and, second, how their computational performance compares with one another.

Table I: The three experimental setups (*E1*, *E2* and *E3*) and their performance numbers. The ranks for simulation tasks are mapped to cores while the ranks for training tasks are mapped to nodes (each Theta node consists of CPU 64 cores).

	E1	*E2*	*E3*
# Total cubic sample	600	600	3000
# Total trigonal sample	11520	11520	57600
# Total tetragonal sample	20000	20000	100000
# Epochs	100	40	250
# Rank for simulation	256	256	512
# Rank for training	4	4	16
# Phase (serial/parallel workflow)	2	2	4
Running time, T_b, for $\mathcal{W}_b$ (sec)	2720	1900	8633
Running time, T_s, for $\mathcal{W}_s$ (sec)	2573	1759	8752
Running time, T_p, for $\mathcal{W}_p$ (sec)	2301	1394	7222
Serial-to-baseline speed-up, T_b/T_s	1.057	1.080	0.986
Parallel-to-baseline speed-up, T_b/T_p	1.182	1.363	1.195
Parallel-to-serial speed-up, S_{p2s}	1.118	1.262	1.212

Model Performance: Although the same model architecture is used in the three different workflows, the models ultimately trained are characterized by different model weights because of the differences in how each is trained in the three different workflows.

For the baseline workflow, all the training samples (see Table I) are generated once in the beginning and then the model $\mathcal{M}_b$ is trained for 100 epochs. For the $\mathcal{W}_s$ workflow, the bias-free policy described above was used to separate the data generation into two phases. The model $\mathcal{M}_s$ was then trained using the data generated in phase 1 for 100 epochs, and then the training was continued with the data generated in phase 2 for another 100 epochs. The baseline workflow $\mathcal{W}_b$ can be viewed as a special case of the serial workflow $\mathcal{W}_s$ where the number of phases is one. The model $\mathcal{M}_p$ trained by the parallel workflow $\mathcal{W}_p$ is identically trained as in $\mathcal{W}_s$ with the training task of one phase overlapped with the simulation task of the next phase. The test loss as a function of the number of local batches being trained is plotted in Fig. 3a. In this study, 5% of the total data was used for testing in each case. Fig. 3a and Fig. 3b illustrate two important points. First, the testing loss for $\mathcal{M}_s$ and $\mathcal{M}_p$ are identical. While this is not surprising since $\mathcal{W}_s$ and $\mathcal{W}_p$ are identical (the cyan and black curves are overlapping in Fig. 3a and Fig. 3b) in the computations performed except that the simulation and training tasks overlap in one compared to the other. The second and more important point is that the testing loss of both models, $\mathcal{M}_s$ and $\mathcal{M}_p$, are comparable to that of the baseline model $\mathcal{M}_b$ and, in fact, converge asymptotically in the number of local batches in both experiments *E1* and *E3*. This suggests that the multi-phase parallel workflow will not compromise the accuracy of the baseline workflow. The test loss slightly increases right after 1500 local batches. This is where the training in the second phase starts, and the reason for the slight increase in the test loss can be attributed to the fact that the model trained in the first phase does not generalize immediately.

Computational Performance: In Table I, the runtimes of each experiment for the indicated number of epochs are denoted by T_b, T_s and T_p corresponding to the workflows $\mathcal{W}_b$, $\mathcal{W}_s$ and $\mathcal{W}_p$, respectively. We compare the computational speedup, if any, from the workflows $\mathcal{W}_s$ and $\mathcal{W}_p$ by comparing their corresponding runtimes with the runtime of the baseline workflow, $\mathcal{W}_b$, respectively. Accordingly, the speedups from $\mathcal{W}_s$ and $\mathcal{W}_p$ are T_s/T_b and T_p/T_b, respectively.

In *E1*, the serial-to-baseline speedup is found to be 1.057 and the parallel-to-baseline speedup is found to be 1.182. In addition, the parallel-to-serial speedup is found to be 1.118. These speedup factors are small due to the imbalance between the training and simulation tasks - the running time of the training task was about three times more than that of the simulation task.

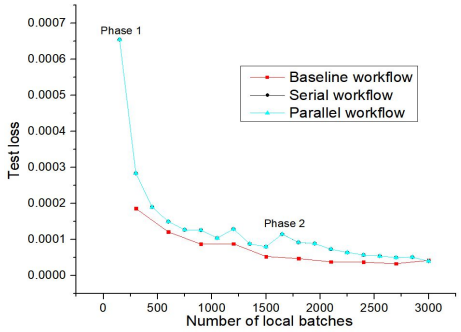
If it is assumed that each simulation sub-task takes approximately the same time t_s, and each training sub-task takes approximately the same time t_t, then the theoretical speed-up of the parallel workflow with respect to the serial workflow is:

$$\text{Speedup, } S_{p2s} = \frac{n \cdot (t_s + t_t)}{t_s + t_t + (n-1) \cdot \max(t_s, t_t)} \qquad (1)$$

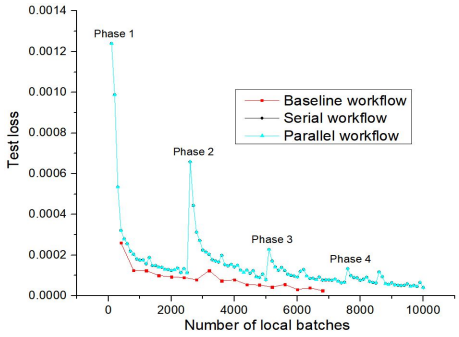
This speed-up is expected to increase as t_t and t_s approach each other, suggesting that when the simulation and training sub-tasks are designed to roughly consume equal amounts of time, the S_{p2s} speedup is expected to improve. This improvement is also expected with increasing number of phases n.

In *E2*, which is identical to *E1*, the model is trained for only 40 epochs in each phase. Since the model does not converge in *E2* due to the reduced number of epochs, the corresponding test losses are not presented here. However, the $S_{p2s} = 1.262$ in *E2* is better than that of *E1* and is close to the theoretical maximum speedup of 1.333 when $n = 2$, according to Eq. (1). The two main reasons why the theoretical limit was not achieved are: (a) the training time and the simulation time are not balanced, and (b) RCT incurs some task scheduling overhead.

In *E3*, five times greater training data is generated than in *E1* or *E2*. Also, more resources are used for simulation and training. See Table I. Additionally, the model was trained for 250 epochs instead of 100, which made it an imbalanced scenario. As expected, the observed S_{p2s} speedup of 1.212 was smaller than the theoretical value of 1.6 when $n = 4$ due to this imbalance. The test loss as a function of the number of local batches being trained, plotted in Fig. 3b, shows that the final test loss in the parallel workflow is slightly higher than that of the baseline workflow. Note the three spikes in the test loss at the advent of a new phase. These spikes gradually decrease in the latter phases as the model generalization improves.



(a) *E1*



(b) *E3*

Figure 3: Loss function on the test set as a function of the number of local batches trained. The loss for the serial and parallel workflows are identical and, hence, overlap.

VII. Summary and Future Work

This short paper reports 20% improvements in computational speed without any appreciable loss of model accuracy resulting from modifications in a typical data generation and model training workflow for neutron scattering data analysis. In the present study, the ML models in every phase of the workflows were trained to the same number of epochs. In future, the stopping criterion will be changed to an early stop condition in which the models will stop training as soon as the calculated loss over a pre-defined number of consecutive epochs remains bounded by some pre-defined loss value. In addition, active learning policies will be integrated into the workflows to guide the task of generating lesser, but carefully chosen, training data to further reduce simulation and training times compared to the baseline workflow.

References

[1] C. Garcia-Cardona, R. Kannan, T. Johnston, T. Proffen, and S. K. Seal, "Structure Prediction from Neutron Scattering Profiles: A Data Sciences Approach," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 1147–1155.

[2] B. H. Toby and R. B. Von Dreele, "GSAS-II: The Genesis of a Modern Open-Source All Purpose Crystallography Software Package," *Journal of Applied Crystallography*, vol. 46, no. 2, pp. 544–549, 2013.

[3] V. Balasubramanian, S. Jha, A. Merzky, and M. Turilli, "RADICAL-Cybertools: Middleware Building Blocks for Scalable Science," 2019. [Online]. Available: https://arxiv.org/abs/1904.03085