Using Deep Learning for Automated Communication Pattern Characterization: Little Steps and Big Challenges

Philip C. Roth
Computer Science and Mathematics Department
Oak Ridge National Laboratory
Oak Ridge, TN USA
rothpc@ornl.gov

Kevin Huck
Department of Computer and Information Science
University of Oregon
Eugene, OR USA
khuck@cs.uorgeon.edu

Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, UT USA
ganesh@cs.utah.edu

Felix Wolf
Department of Computer Science
Technische Universität Darmstadt
Darmstadt, Hesse, Germany
wolf@cs.tu-darmstadt.de

***Abstract*—Characterization of a parallel application's communication patterns can be useful for performance analysis, debugging, and system design. However, obtaining and interpreting a characterization can be difficult. AChax implements an approach that uses search and a library of known communication patterns to automatically characterize communication patterns. Our approach has some limitations that reduce its effectiveness for the patterns and pattern combinations used by some real-world applications. By viewing AChax's pattern recognition problem as an image recognition problem, it may be possible to use deep learning to address these limitations. In this position paper, we present our current ideas regarding the benefits and challenges of integrating deep learning into AChax and our conclusion that a hybrid approach combining deep learning classification, regression, and the existing AChax approach may be the best long-term solution to the problem of parameterizing recognized communication patterns.**

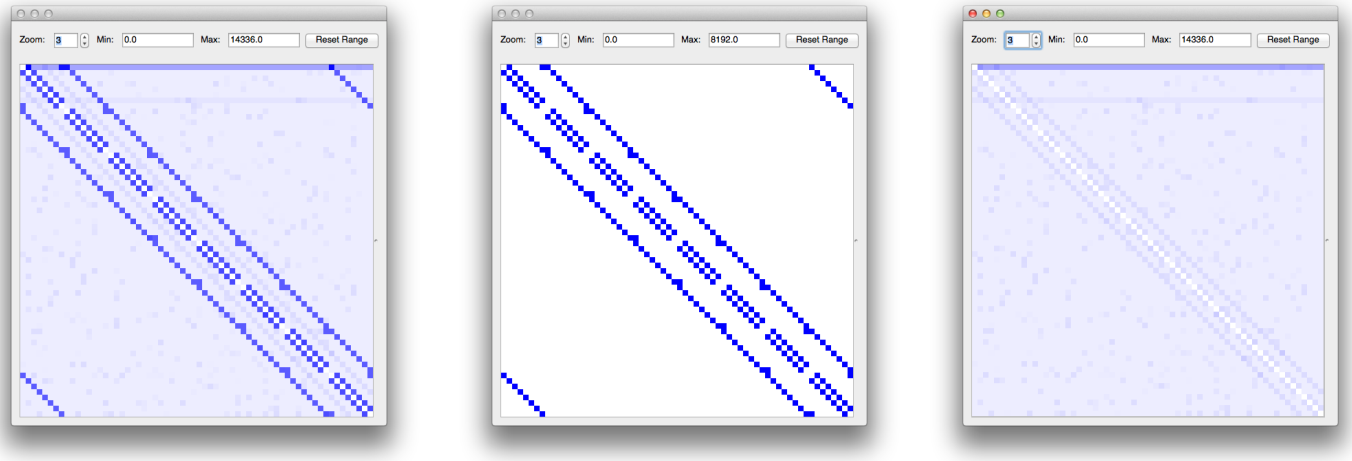
***Index Terms*—deep learning, automation, communication, characterization**

I. Introduction

Over the past few years, one of us (Roth) has been developing an approach for automatically recognizing and characterizing the communication patterns of parallel applications [1], [2]. The approach uses search and a library of known communication patterns like Broadcast and 3D Nearest Neighbor. The input to the approach is a representation of a parallel application's communication behavior. Logically, this information is represented as an Augmented Communication Graph [2] (ACG), a graph that captures the volume and operation count of the collective and point-to-point communication operations performed by each process during an application run. At each step of its search, the approach examines the communication data that has yet to be explained (called the *residual*) to see if it can recognize any communication patterns from its pattern library. If it recognizes a pattern in a residual, it determines the parameters of the pattern (such as its *scale*, the amount of data that was transferred in the operation) and then refines its search by removing the contribution of the parameterized pattern to form a new residual, from which it continues its search. Figure 1 demonstrates this recognize-parameterize-remove operation. Because the approach might recognize multiple patterns within a residual, the search results form a tree where each path from the tree's root to its leaves represent a collection of parameterized patterns that have been recognized in the original communications data. The path whose leaf has the smallest residual represents the collection of patterns that best explains the original communications data. By reporting the name and parameters of each pattern along this path, the approach generates a concise description of the application's communication behavior that is easier to manage than a full communications event trace and more accurate than summary statistics.

The approach has a few known limitations. One important limitation is that it does a poor job of handling patterns where the amount of data transferred between senders and receivers may vary, such as a nearest-neighbor pattern used in a molecular dynamics simulation. Although we have explored heuristic techniques for determining a pattern's scale that avoid trapping the search in local search space minima [2], our recognition implementation still assumes that the amount

This manuscript has been co-authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE 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) Example communication residual. (b) 2D nearest neighbor pattern. (c) Resulting communication residual.

Fig. 1: Removing a recognized, parameterized communication pattern from an example residual (visualized as a traditional communication matrix). Screen captures originally presented in [1].

of data transferred in a pattern does not depend on the particular sender and receiver and thus may fail to explain all of the application's observed communication behavior if this assumption is not true.

AChax is a Python-based tool that implements this automated communication characterization approach for applications that use a Message Passing Interface [3] (MPI) implementation for communication and synchronization. The tool's distribution includes a library that provides interposition functions for many MPI communication calls made by an application as it runs, and outputs an ACG that captures the application's MPI communication behavior. After a brief dalliance with using graphs built using the Graph-tool Python module [4] as an internal ACG representation, the tool once again represents ACGs using an adjacency matrix encoded in a NumPy [5] matrix because the tool's analysis performance is much better using matrices than when using the Graph-tool-based ACG representation.

A presentation at a recent tools workshop describing AChax [6] spurred us to form an informal working group techniques and challenges with automated pattern recognition in performance, debugging, and characterization tools. Although our discussion ranged widely, the AChax pattern recognition challenges turned out to be the dominant topic. We have long known that we can view the AChax pattern recognition problem as an image recognition problem. (Indeed, capturing the human expertise required to recognize patterns within visualizations of communications adjacency matrices is the primary motivation for the AChax work.) Because of deep learning's well-demonstrated capability for automatic image classification, including images that are "fuzzy" or otherwise obfuscated, deep learning seems tailor-made for the AChax communication pattern recognition problem and we spent a significant part of our working group discussion on exploring the potential benefits and challenges of its use in the AChax context.

In this position paper, we capture the gist of our workshop discussion, and add more detail and perspective based on subsequent consideration and hands-on experimentation using deep learning for the AChax image recognition problem. We describe how we might use our current AChax implementation to train a model using a deep neural network (DNN) and how we might use that model for communication pattern recognition. We discuss the challenges of using a model for parameterizing a recognized pattern. And we present our very early experience with training and using a model to recognize some of the patterns from AChax's current pattern library that lead us to propose that a hybrid strategy combining deep learning with our traditional recognition approach may be the best option for a future AChax implementation. It is also worth noting that at least some of us are *not* deep learning experts and are approaching this study to establish whether the proposed approach is feasible enough to warrant further investigation that includes team members with stronger deep learning expertise.

II. Integrating Deep Learning into AChax

At first blush, the integration of deep learning into our existing communication pattern characterization approach seems like an easy prospect. From a high-enough conceptual level, it seems as simple as replacing our current pattern recognition approach with one that feeds a residual matrix into a model trained to recognize the patterns from our existing library. From a practical perspective, because AChax is implemented using Python and because several of the common deep learning implementations such as TensorFlow [7], Theano [8], and PyTORCH [9] provide well-documented Python interfaces, it should be relatively easy to make use of one of these

frameworks in our current AChax software. Nevertheless, considering the details reveals several significant challenges to be overcome.

A. Training

A model's DNN must be trained to recognize the patterns from the AChax pattern library. AChax's current implementation eases this training activity, because each pattern in AChax's pattern library is implemented as a Python class that implements both a *generator* and *recognizer* method.[1] A pattern's generator method takes a collection of parameters meaningful to the pattern (such as the dimensions of a 3D nearest neighbor pattern), and generates a matrix representing the parameterized ACG of that pattern. This "pure" matrix is used by some patterns as a mask during the pattern recognition step, and by all patterns when removing the recognized pattern from a residual.

A version of AChax that uses deep learning could use these generated matrices to produce training data. How best to label that data remains an open question. At a minimum, the label could include only the pattern's name, in which case we expect the resulting model to be useful only for identifying the type of pattern that is most strongly represented in the input residual matrix. Some other method would be needed to determine the pattern's parameters (e.g., the approach currently used within AChax). Although this approach might seem to add little value over the existing AChax approach, we believe it could be a necessary part of adding the ability to recognize patterns with varying amounts of data transferred between source and destination processes. At the other end of the spectrum, we could include the pattern's name and *all* of the parameters used to generate the training matrix in its training set label. This approach would likely result in an unfeasible number of classification categories, and we suspect that this level of specification would result in a model that is overfitted to the training data.

The sweet spot is likely to be somewhere between these extremes, leading to a model that can identify not only the pattern's name but also *some* information about its parameters that would accelerate the AChax recognizer's ability to determine the complete parameterization. For instance, it may be the case that a trained model can recognize the dimensions of a 2D or 3D nearest neighbor pattern, or from which side or corner a sweep pattern originates. It may also be beneficial to use a two-phase approach whose first phase involves classification of the basic pattern, and whose second phase attempts to discriminate between the specific alternatives that might be present for that basic pattern. We discuss a few more aspects of parameterization in Section II-B.

In addition to these questions of *how* to train a model to support identification of a pattern's parameters, there is also a question of *when* to do this training. Because we would be training our model with ACG matrices representing "pure" patterns, it might be an appealing idea to pre-generate an application-independent library of trained models for process counts commonly used in application runs (e.g., all powers of two between 16 and 16384). In practice, however, we expect this general-purpose library of trained models to be of limited use: by definition, it would not support applications for which non-power-of-two process counts are the best choice, and it would not support applications that subdivide their processes into smaller groups and communicate within these subgroups (e.g., using MPI sub-communicators). Instead, it seems more likely that a deep learning-based AChax would train its model on demand when invoked with a specific ACG matrix, though it may be possible to save its trained model to an *application-specific* model library.

[1]The Garbage pattern is an exception: it only provides a generator method because this pattern's only purpose is to introduce "noise" into synthetic workloads used in unit testing.

B. Recognition and Parameterization

Applying the trained model to a residual matrix results in a vector of probabilities P, one per training category, such that the probability of the residual containing training specification category i is P_i. If one of these probabilities is much larger than the others, the model has given clear indication that the associated training category is highly likely to be present in the residual. But if several probabilities are nearly equal, the meaning is less clear. If those probabilities are large, we would interpret the model's output as indicating the patterns are present in the residual at nearly equivalent scales. In this case, AChax would refine its search along each of the patterns and rely on its ability to eventually distinguish between the quality of the resulting search paths once its search is done. On the other hand, if the probabilities are small, we assume that patterns from the associated training categories are not present and the search can be pruned at that point.

As noted above, there are many open questions regarding use of deep learning for parameterization of recognized patterns, and using classification can take us only so far with respect to parameterization. We expect that some parameters will require us to use a regression model instead. In particular, we expect to need regression to predict the scale of a recognized pattern. The scale indicates how much data was transferred between source and destination processes during the communication operation. It remains to be seen whether using regression to estimate the pattern's scale outweighs the accuracy of AChax's current approach of examining each of the values associated with the recognized pattern within the residual and setting the scale based on those values (e.g., their maximum or average), but the regression approach may prove to be more useful for patterns with varying amounts of data transferred between source and destination processes.

III. Early Experiments

As a first step in determining whether it is both feasible and useful to incorporate deep learning into AChax, we conducted a few simple experiments to determine whether we could train a model to recognize several of the basic patterns from the existing AChax pattern library. We conducted our experiments

using TensorFlow 1.10.1, Python 3.6, and a development version of AChax from the "acg-matrix" branch of its repository. Because we were more concerned with the trained model's accuracy than its performance, we ran the experiments on a Mac OS X laptop that already had the required software stack to run TensorFlow models. For all experiments, we constructed models for a hypothetical application that was run with 256 MPI processes.

In our simplest experiment, we constructed 1000 images, each of which represented a "pure" Broadcast or Reduce pattern with randomly-selected root process, or 2D 5-point Nearest Neighbor pattern, each with randomly selected scale. We used 950 images to train our model, and 50 to test its accuracy. With this simple training/testing set, the model reached close to 100% accuracy in five training epochs, but still achieved 100% accuracy on its training images. Adding noise to the training and testing images caused a slight decrease in the model's training accuracy, but it still achieved nearly 100% accuracy with its training set.

Although the ability to recognize a single communication pattern from a (possibly noisy) image is a necessary capability for use within AChax, it is hardly sufficient. Rather, AChax needs the ability to recognize communication patterns in images with multiple patterns. To test this capability, we trained a model as described above, and used it to predict the likelihood of presence of its known patterns in a test set of 5 images, each containing all three communication patterns, with noise. Figure 2 shows an example of one of these multi-pattern images. For each of the five images, with or without noise, the trained model predicted the image contained one of the three patterns with 100% confidence. From an AChax perspective, this may be a desirable behavior because it allows the tool to easily choose which pattern remove next, we mistakenly expected the model to output priorities that reflected each pattern's degree of "presence" within the image as determined by each pattern's scale. We assume that the model chose exactly one pattern in each of our test matrices because each of our training matrices contained only one pattern, and we assume that we would have to train using matrices representing combinations of patterns to obtain the prediction behavior we originally expected.

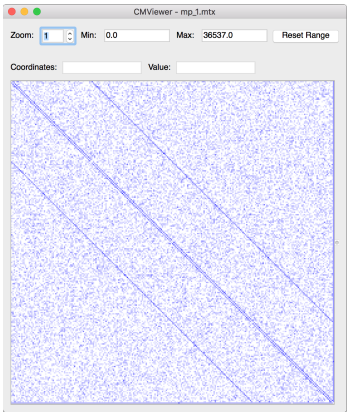


Fig. 2: Example multi-pattern image used to test our trained deep learning model.

IV. Summary

Deep learning seems tailor-made for the pattern recognition problem of the AChax automated communication pattern recognition tool. It seems especially attractive for addressing the current AChax limitation of being unable to completely account for the communication from patterns where the amount of data transferred depends on the specific source and destination processes. In this position paper, we discussed our current ideas about how deep learning might be integrated into the AChax search-based communication pattern recognition approach, the challenges of doing so, and some very early experiences in using a trained model to recognize synthetic communication patterns generated by the current AChax implementation. Our experience indicates that using a trained deep learning model to recognize patterns is feasible, but may require both classification and regression, or a hybrid approach combining deep learning with our existing parameterization techniques to identify the full parameter set to associate with recognized patterns.

Acknowledgments

We thank David Poliakoff of Lawrence Livermore National Laboratory for his helpful feedback about this paper and the tools workshop presentation that motivated it.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under contract number DE-AC05-00OR22725.

This work is supported in part by the US Department of Energy Office of Science SciDAC RAPIDS project under subcontract 4000159855 to the University of Oregon from Oak Ridge National Laboratory.

References

[1] P. C. Roth, J. S. Meredith, and J. S. Vetter, "Automated characterization of parallel application communication patterns," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC'15)*, Portland, Oregon, USA, Aug 2015, pp. 73–84.

[2] P. C. Roth, "Improved accuracy for automated communication pattern characterization using communication graphs and aggressive search space pruning," in *Proceedings of the 6th Workshop on Extreme-Scale Programming Tools (ESPT'17),* to be published as Lecture Notes in Computer Science **11027**, 2018.

[3] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*, 2nd ed., ser. Scientific and engineering computation. Cambridge, MA: MIT Press, 1999.

[4] "Graph-tool: Efficient network analysis," https://graph-tool.skewed.de, 2018.

[5] "NumPy," http://www.numpy.org, 2018.

[6] P. C. Roth, "Scalable, automated characterization of parallel application communication behavior," in *2018 Scalable Tools Workshop*, Jul 2018.

[7] M. Abadi, A. Agarwal *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2015. [Online]. Available: http://download.tensorflow.org/paper/whitepaper2015.pdf

[8] R. Al-Rfou, G. Alain *et al.*, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[9] A. Paszke, S. Gross *et al.*, "Automatic differentiation in PyTorch," in *NIPS 2017 Autodiff Workshop*, Dec 2017.