openDIEL: A Parallel Workflow Engine and Data Analytics Framework

Frank Betancourt
The University of Tennessee
fbetanco@vols.utk.edu

Kwai Wong
The University of Tennessee
kwong@utk.edu

Efosa Asemota
Morehouse College
efosaasemota@gmail.com

Quindell Marshall
Morehouse College
quindell.marshall@morehouse.edu

Daniel Nichols
The University of Tennessee
dnicho22@vols.utk.edu

Stanimire Tomov
The University of Tennessee
tomov@eecs.utk.edu

ABSTRACT

openDIEL is a workflow engine that aims to give researchers and users of HPC an efficient way to coordinate, organize, and interconnect many disparate modules of computation in order to effectively utilize and allocate HPC resources [13]. A GUI has been developed to aid in creating workflows, and allows for the specification of data science jobs, including specification neural network architectures, data processing, and hyperparameter tuning. Existing machine learning tools can be readily used in the openDIEL, allowing for easy experimentation with various models and approaches.

CCS CONCEPTS

• **Software and its engineering** → **Software creation and management**; **Designing software**; **Software design engineering**;

KEYWORDS

workflow engine, neural network, data science, graphical user interface

1 INTRODUCTION

When conducting complicated simulations, it is often required to utilize a large variety of applications to answer a research question. Utilizing disparate modules of computation, requires the careful coordination of dependencies, communication, and synchronization between them, and there is not always a clear path on how to do these kinds of tasks. One approach to this is to utilize a script to run all of the necessary modules. However, this falls short, as this loses much detail with regard to complex interaction between modules, such as inter-moduler communication and non-linear dependencies, limiting the ability to take full advantage of HPC systems to run modules in parallel.

This problem is solved by workflow engines, allowing researchers to define complex dependencies between modules, and schedule communication between them. openDIEL specifically focuses on unifying modules into a single executable that uses MPI for communication. Additionally, methods are provided for specifying dependencies between modules, with the framework resolving dependencies and running modules in parallel.

openDIEL is designed not only to support the creation of generic workflows, but facilities are also provided to create data science workflows, supporting simple data cleaning, creation of neural network architectures, and subsequently searching for optimal parameters with a grid engine.

The major work that has been done to extend the original openDIEL framework is to provide a GUI with the goal of increasing usability. The standard manner in which workflows are configured is to edit configuration files that define the way the engine will work. The major drawback to this approach is that it is not always clear how to set up a file or even what options are available. The GUI overcomes this by providing a clear list of what can be specified, which reduces the load on the end user to think of everything ahead of time, but instead recognize what needs to be done.

2 RELATED WORK

Existing workflow engines such as Galaxy give researchers the means to utilize tools on cloud computing resources. Galaxy is designed to operate with and couple together domain-specific tools, without requiring the end user to do large amounts of work to integrate tools together [1]. Apache Airavata is another workflow engine that is designed to manage complex computational jobs and workflows. It is designed to work on a variety of different computing resources, including local clusters and the cloud [8].

openDIEL shares many of the same goals core goals as these frameworks, but differs in several ways. The first is that the openDIEL framework uses MPI as a core component for communication between modules and for managing running workflows. The other way in which openDIEL differs is that it is designed specifically for tightly coupled HPC systems.

Another one of the goals of the openDIEL GUI is to provide a general framework for performing hyperparameter search and model selection tasks. A number of tools already exist to assist in doing this. Tools such as HyperBand, Optunity, and BayesOpt provide methods to perform optimization with a variety of different algorithms, but do have support for training in a distributed manner out-of-the-box [3, 6, 9]. Many existing systems also lack support for implementing algorithms beyond what is provided in the tool kit. The Tune Framework is a system that supports custom algorithm implementation, and is designed to be used in a distributed manner [7].

openDIEL Framework Design Overview

At a high level, the openDIEL system consists of a C library that contains all of the function needed to manage modules. Typically, a main driver file is created which contains all of the needed function calls to set up the main MPI communicator that the IEL library uses, set up necessary modules for tuple space communication, and calling the user defined modules.

The main way that users interact with the system is through a configuration file. There are two major components of the openDIEL system: the executive library, and the communication library.

Configuration File. Information about how modules are communicate and rely on one another is contained in a configuration file. The configuration file defines what resources the modules requires, such as the number of cores, and number of GPUs required by the module. After defining the modules themselves, a section of the file subsequently defines the manner in which groups of modules depend on one another, how many iterations need to run, amongst other characteristics.
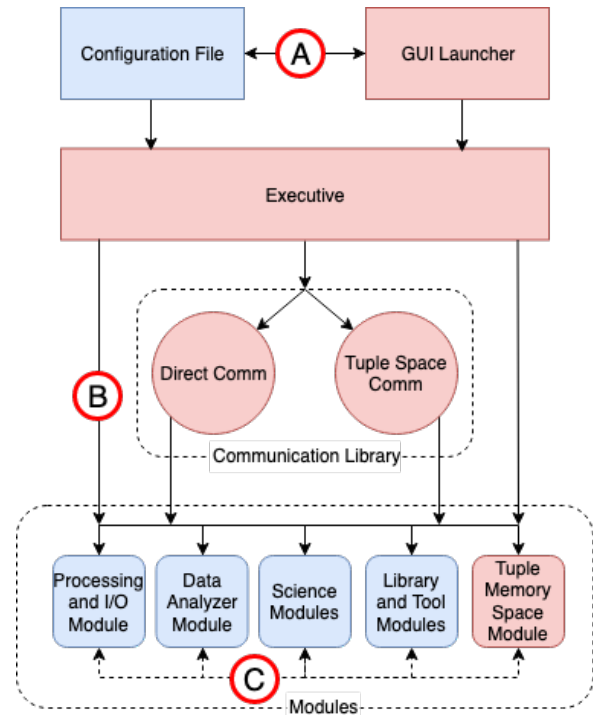

Figure 1: Overview of architecture of openDIEL, top to bottom: (A) The GUI launcher creates a configuration file for the workflow, and the executive will read this file to set up workflows. (B) After initial configuration, the executive will start all of the modules. (C) The modules have access to the communication library, and directly communicate or utilize tuple space communication.

Communication Library. The communication library is essentially a wrapper around various openMPI function calls, and is responsible for managing both tuple space and direct communication between modules. This is done by creating a main MPI_COMM_WORLD communicator in which all of the modules run, and then subdividing this main communicator at the level of single modules. If there are multiple concurrent copies of the same module running at the same time, then the module sub-communicator is further subdivided between the copies [13].

Two different methods of communication are provided by an API: tuple space communication, and direct module-to-module communication. With tuple space communication, a tuple server module is utilized that allows modules to concurrently send data to and receive data from a shared associative array. Modules can use this form of communication with provided library functions IEL_tput and IEL_tget to send and receive data from the tuple space respectively. Each module that puts data into the tuple space can issue a non-blocking IEL_tput function call, and provide a tag for the data placed in

the tuple space. The receiving module can utilize a blocking IEL_tget function call to retrieve the data with the specified tag.

The IEL_tput function is a wrapper around the MPI_Send function, and the tuple server runs as an MPI process in the MPI_COMM_WORLD communicator. The tuple server receives the data from a module, and then places it into a red-black tree keyed on the tag, and values is a queue into which the data is placed. The IEL_tget function utilizes the MPI_Send function to notify the tuple server that it wants to retrieve a tag, and whether or not it wants to remove the data from the tuple server after retrieving it. MPI_Recv is then called, and the data is retrieved.

Simplified versions of IEL_tput and IEL_tget are also provided: IEL_tput_simplified and IEL_tget_simplified. The allow users to specify the name of the calling module and tag for the data as a string, and does not require the user to specify the rank upon which the tuple server is running. This function is just a wrapper around the normal IEL_tput function call, but the string provided is hashed and used as the tag in the call.
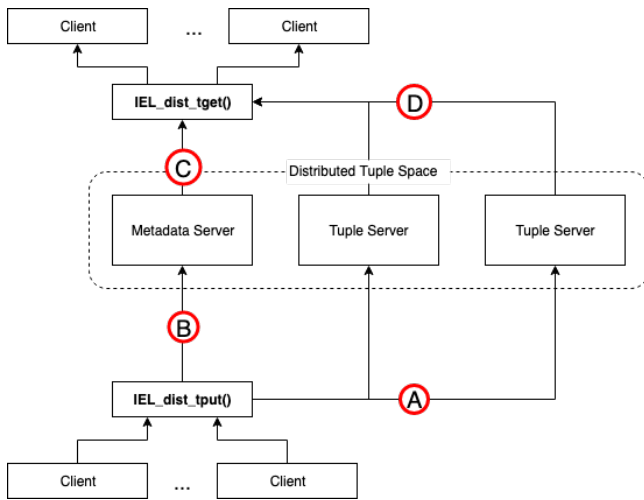

Figure 2: Layout of the distributed tuple server model, clockwise from the bottom right: (A) The client data is broken up and distributed across an array of tuple servers, and (B) the metadata for the distributed data is stored in a metadata server. (C) When data is to be retrieved, the metadata for the requested data is retrieved, and (D) the data is retrieved from the distributed array of tuple servers based on the metadata. The received data is then reconstructed and returned to the requesting client.

The tuple space can also be distributed across a number of different modules, essentially providing a way to store and retrieve data in a distributed manner. The functions IEL_dist_tget and IEL_dist_tput utilize this multiple tuple

server model. The IEL_dist_tget function will take a pointer to data and a string to tag the data, and distribute it amongst an array of tuple servers. Information about the distribution of the data is stored on a meta-data server. The IEL_dist_tget function will retrieve the data by querying the meta data server, which returns the locations of the servers holding the data, the data servers are queried, and the stored data is reconstructed.

Executive Library. The executive library is the other major part of the library responsible for starting job and managing dependencies. When a job starts, the executive will read in a workflow configuration file, and then based on this file, the executive will create a dependency graph of the specified workflow, and then start modules based on the graph. Typically, a module is included in openDIEL by linking a library against a driver file, and function pointers are provided to the executive so that they can be called with the appropriate arguments [13]. Executables can also be run by calling fork() and exec() in an MPI process, but limits the ability of the module to utilize the inter-modular communication provided by openDIEL.

Typical Usage. Typically, all of the needed functions are called in a main driver file. This driver file will call MPI_init, and then it will call openDIEL member function IELAddModule, which will take a pointer to the function in the linked library for the module. This will be used later to start the module in the workflow. For modules that are executables, a model that calls fork() and exec() on the proper arguments is started for each serial module. After this setup, the main IELExecutive() member function is called. This function will split up the MPI_COMM_WORLD communicator into the appropriate subcommunicators, resolve dependencies from the configuration file, and then start modules.

Graphical User Interface

The GUI provides methods to easily specify modules, the resources required to run them, and the libraries needed to link against in order to run them. For workflow specification, methods to visually display the resulting workflow as a dependency graph are provided to aid in the ease of creating a workflow. The other enhancement is automatically creating a driver file to run parallel codes. A number of preconfigured modules can also be provided, allowing for a reduced amount of set up on the part of the end user of the system.

While manual configuration can still be accomplished, the graphical user interface aims to reduce the amount of work required to set up a workflow, and improve the overall use ability of the openDIEL framework. Most operations the GUI are normally accomplished with editing configuration files, writing code in a driver file, linking libraries, and writing

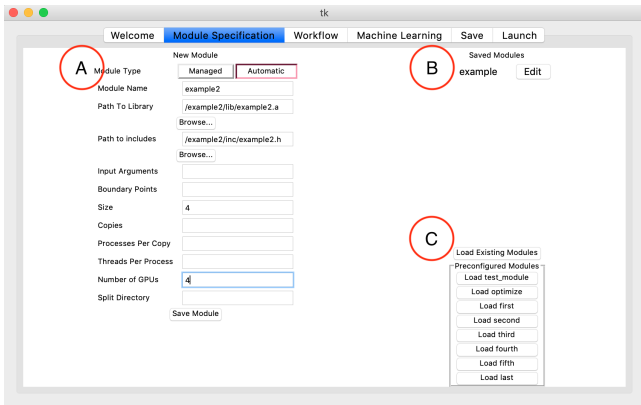modules. The GUI itself is divided into several different tabs:

Figure 3: The module specification tab, clockwise from top left: (A) This section is where new modules are created, and where created modules are edited. (B) A list of the saved modules. (C) A listing of the preconfigured modules which can be loaded, along with the option to load modules previously created by the user

Module Specification. The module specification tab allows users to specify information about each individual module. Resource requirements such as the number of processes needed (*size*), the number of copies of the module to run, and the number of GPUs needed can be specified, among others. Additionally, a number of preconfigured modules can be loaded. These preconfigured modules consist of all the basic information needed to include it in the openDIEL system, such as library locations, function calls, and basic resource requirements.

Workflow. After configuring modules in the workflow tab, these modules can then be added to workflows. Modules are organized into groups, and dependencies are defined between groups of modules. Within groups, a list of modules is specified that is run linearly. If no other groups are specified as dependencies, the groups will run as resources become available, making parallelism the default. Workflows can also be loaded from files, which allows users to use workflows that have already been created, and save workflows for later use.

Save and Launch. The Save and Launch tabs are used for saving the current state and launching the job respectively. The save tab will create a configuration file that can be either used by the GUI later on to load in the same workflow, or it can be used by the openDIEL back end to run a workflow job. The launch tab allows a user to launch the job as it was previously specified, and directs the output to a specified
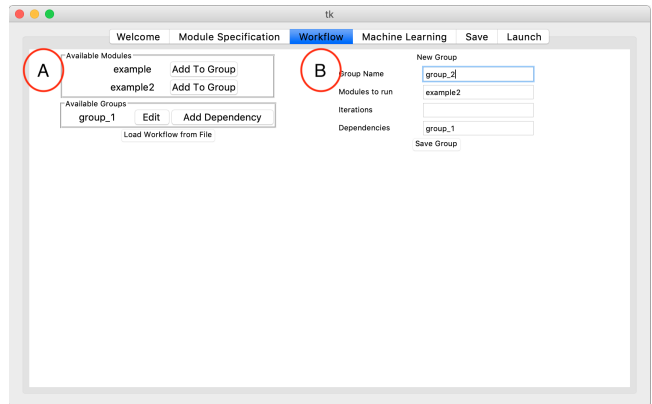

Figure 4: The workflow specification tab, left to right: (A) Listing of the modules defined in the previous section of the GUI, and a list of the already created workflow groups. This is used to add modules to workflow groups, and to add an existing workflow group as a dependency to the current group. (B) Entry fields for the group currently being defined.
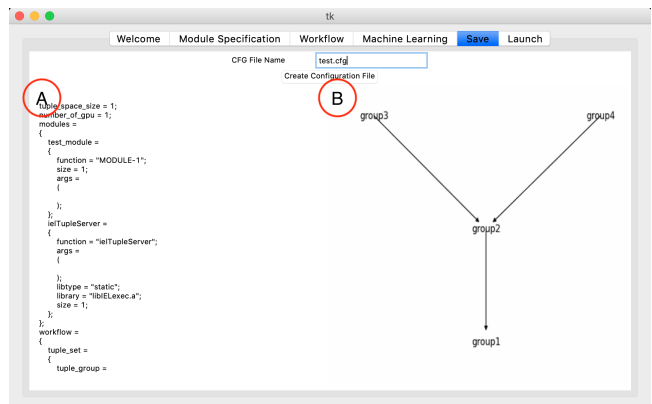

Figure 5: The Save Tab, left to right: (A) The contents of the workflow configuration file that was created and saved. This is what the driver will read when it starts in order to configure modules. (B) A dependency graph of the groups to aid in visualizing the dependencies between configured workflow groups. The vertices represent workflow groups, and directed edges terminate at the group the originating vertex depends upon.

directory. Example workflows are also provided for the purposes of tutorials, which will load a pre-existing workflow containing the requisite information to run.

Machine Learning. The Machine Learning tab implements a number of functions. Users can specify a CSV data file, and get a basic idea of the contents of the file, such as the number of rows, columns, and the names of the columns in the file. The option to perform common pre-processing tasks on columns in the data set is provided, such as data
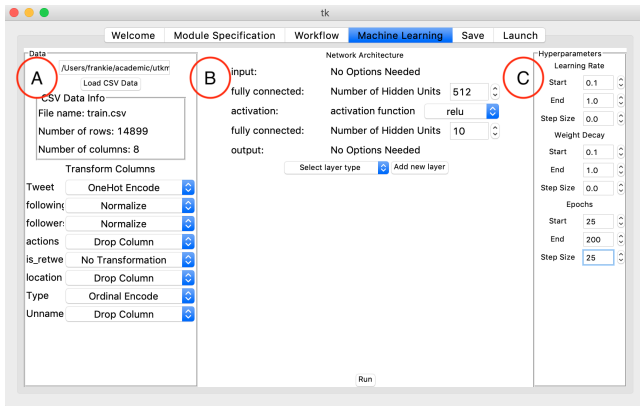
Figure 6: Machine Learning Tab, left to right: (A) Data tab, CSV files can be loaded, and a summary of the columns is provided. Shown are a number of transformations that can be applied to the columns of the CSV file that was read in. (B) Network architecture can be defined here, along with the parameters for each layer. (C) Hyperparameters grid search range parameters can be defined here.

centering, normalizing, missing value imputation, and encoding. The interface also provides a method to specify a DNN architecture for classification tasks, with support for fully connected, flattening and activation layers. After specify the model, hyperparameter search can be performed. This is done by providing a starting, and ending value for the hyperparameter, and a step size. The grid search is then performed on the Cartesian product of all of the parameters.

Grid Engine

One of the goals of the framework is to not only provide facilities for hyperparameter optimization and search, but allow for users to readily use existing libraries to perform these tasks. One such implementation is a grid search engine that uses the openDIEL tuple space communication to distribute parameters to worker processes in an exhaustive search of a specified parameter space, collect the results, and report the best parameters found.

The module consists of a master process that chooses hyperparameters, and a set of processes that receive the hyperparameters. The master process first selects a set of parameters, distributes them to the workers via the tuple space, and waits for the group to finish. The group of worker processes receive the parameters, train, and report their results to the trainer via tuple space communication. These results are then gathered by the master process, and then the next group of processes is started on the next batch of parameters.

openDIEL uses magmaDNN to define neural networks. magmaDNN is a deep neural network library that makes use
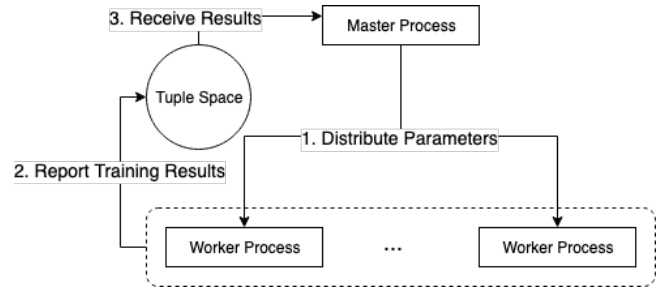

Figure 7: Distributing Parameters to Workers With Tuple Space

of highly optimized Magma BLAS to achieve speeds beyond other frameworks [10]. The grid engine module works by creating a configuration file that contains a specification of a parameter space, and a network architecture. When the openDIEL framework starts, this file is read in, and the exhaustive search over the specified hyperparameter space begins. Workers use the identical network architecture, with varied parameters for each run.

Usability Improvements

A large part of the usability of the openDIEL system is hampered by the fact that editing and creating configuration files, creating modules, linking the appropriate libraries to satisfy dependencies, and compiling everything together is often a time consuming and error prone task. One of the goals of the interface is to shift the workflow creation process from a recall task into a task where one only has to recognize options when they are seen.

To highlight the differences between the new interface and the traditional way, an example of setting up a workflow is provided here to compare and contrast between the two.

Suppose a user wants to create and run a workflow. For simplicity, the user wants to run two modules: Module 1 uses MPI and produces a set of files as output, and another module, Module 2 that waits for it to finish and then collects the results from the files that were produced. For Module 1, the user would need to create a static library containing the function that they want to run. For the Module 2, we can assume that it is simply a shell script that gets run after the job is completed.

After both of the modules have been created, a main driver file needs to be created to start the openDIEL system, add function pointers for managed modules, include header files, and appropriately start automatic modules. This often involves using numerous openDIEL member functions, and requires at least a base knowledge of the system in order to correctly create a driver file. After that, the user will then need to compile and link the driver with the static libraries that represent modules, and then it will be in a runnable state.

This process differs somewhat when using the GUI, where the user will provide path to the module library and it's corresponding header files, and then the name of the function is also provided. The GUI then automatically generates the driver file, and compiles it with the provided modules.

With driver file done, the user will then need to create the workflow itself, and define dependencies between modules. With the original method, this would require the user to create a new configuration file, and add the appropriate options. In the workflow file, the user would need to add the appropriate keywords for the number of processes each module requires, along with the function name, amongst other attributes. This requires the user to have knowledge of what these keywords are, and adds additional overhead in terms of what the user needs to know. The GUI eases this burden by automatically generating workflow files with the appropriate formatting and keywords based on the user provides in the GUI. At this point, with the driver file created, and the workflow configured, the workflow can then be started.

3 CONCLUSION AND FUTURE WORK

The openDIEL GUI enhances usability of the openDIEL system and allows for the creation of complex, useful workflows in an expedient and efficient manner. The task of writing drivers, creating configuration files, and knowing the system in general is greatly simplified. Testing the overall usability of the framework as a whole is a major area for future study. A user study would need to be conducted to determine how easily a user could accomplish common workflow tasks, and how much work it takes to learn the system as a whole.

Increasing the ability of the openDIEL framework to do full pipeline optimization of machine learning systems is envisioned. Currently, the hyperparameter search is constrained to simple grid search through a specified search space, leading to an exponential increase in search time as more parameters are added, limiting the feasibility of doing this kind of analysis. Support of more "smartly" choosing parameters is envisioned, by allowing the master process to perform a specified optimization procedure between receiving parameters from workers and distributing new ones. Common approaches such as Bayesian Optimization [12] and learning curve prediction [5] will be supported [2]. Parallel to improving the hyperparameter grid search, support for searching for an entire model is envisioned. Similar to providing the framework with a generic search space for selecting the best hyperparameters, the framework could provide the capability a search for a model.

Currently, magmaDNN is the only way to create machine learning models. In the future, support will be added to use frameworks such as TensorFlow [4], Sklearn [11], and other other machine learning frameworks.

The openDIEL framework provides a powerful set of tools to create workflows with a GUI, and run many different modules on HPC resources. A set of communication primitives is provided to efficiently and easily connect together parallel MPI codes. Machine learning model creation and hyperparameter search is done easily without the need for heavy coding to be done.

4 ACKNOWLEDGEMENTS

REFERENCES

[1] Enis Afgan, Jeremy Goecks, Dannon Baker, Nate Coraor, Anton Nekrutenko, James Taylor, Galaxy Team, et al. 2011. Galaxy: A gateway to tools in e-science. In *Guide to e-Science*. Springer, 145–177.

[2] Tal Ben-Nun and Torsten Hoefler. 2018. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *arXiv preprint arXiv:1802.09941* (2018).

[3] Marc Claesen, Jaak Simm, Dusan Popovic, Yves Moreau, and Bart De Moor. 2014. Easy hyperparameter search using optunity. *arXiv preprint arXiv:1412.1114* (2014).

[4] Sanjay Surendranath Girija. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *Software available from tensorflow. org* (2016).

[5] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. 2016. Learning curve prediction with Bayesian neural networks. (2016).

[6] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2016. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560* (2016).

[7] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. 2018. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118* (2018).

[8] Suresh Marru, Lahiru Gunathilake, Chathura Herath, Patanachai Tangchaisin, Marlon Pierce, Chris Mattmann, Raminder Singh, Thilina Gunarathne, Eran Chinthaka, Ross Gardler, Aleksander Slominski, Ate Douma, Srinath Perera, and Sanjiva Weerawarana. 2011. Apache Airavata: A Framework for Distributed Applications and Computational Workflows. In *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments (GCE '11)*. ACM, New York, NY, USA, 21–28. https://doi.org/10.1145/2110486.2110490

[9] Ruben Martinez-Cantin. 2014. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *The Journal of Machine Learning Research* 15, 1 (2014), 3735–3739.

[10] Lucien Ng, Kwai Wong, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. 2017. Magmadnn high-performance data analytics for manycore gpus and cpus. In *magma-DNN, 2017 Summer Research Experiences for Undergraduate (REU)*.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.

[13] Kwai Wong, Logan Brown, Jason Coan, and David White. 2014. Distributive Interoperable Executive Library (DIEL) for Systems of Multiphysics Simulation. In *2014 15th International Conference on Parallel and Distributed Computing, Applications and Technologies*. IEEE, 49–55.