GUI Development in openDIEL

Omar Tafiti(Morehouse College), Yan Lam, Neptune Wong(City University of Hong Kong), Rocco Febbo(UTK)

Mentor: Dr. Kwai Wong

Table of Contents

1) Introduction

1.1) Research Goal

The goal of the GUI development in openDIEL research project is to develop a fully functional GUI to help create openDIEL modules and run projects using openDIEL.

1.2) openDIEL

1.2.1) What is openDIEL?

The open Distributive Interoperable Executive Library (openDIEL) is a parallel workflow framework. The openDIEL's general purpose at the moment is to run multiple loosely-coupled software together in parallel while providing a useful library of communications functions for data transfer between software. By providing a workflow engine with multiple options which are covered in detail by the examples in the APPLICATIONS directory, the openDIEL allows the user to control the way in which software is run. To run in the openDIEL, the software must be converted to openDIEL "modules," which entails some slight changes to source code (done by the included modMaker script), compiling as a library, and linking at runtime.

1.2.2) How to use openDIEL?

There are a number of steps that the user needs to take to use openDIEL.

1. Create a driver program in C
2. Create a makefile
3. Create modules to be ran
4. Create workflow.cfg file
5. Execute mpirun (calculate number of processes)

OpenDIEL's main component in this entire process is the configuration file. The configuration file has two main sections; The module section and workflow section. The user defines the function modules and schedules its workflow in an input file. The user would then need to determine the number of processes to run the modules. The number of processes can be determined by examining the size of each module and the workflow section. For each workflow group it is the number of processes plus the size of the largest module in each order. Once the number of processes have been determined, the modules are now ready to be ran. These are the steps it takes to run modules using openDIEL. This process can also vary depending on the type of modules the user would like to run. There are some different steps to take for automatic vs managed modules.

1.2.3) Problem with openDIEL

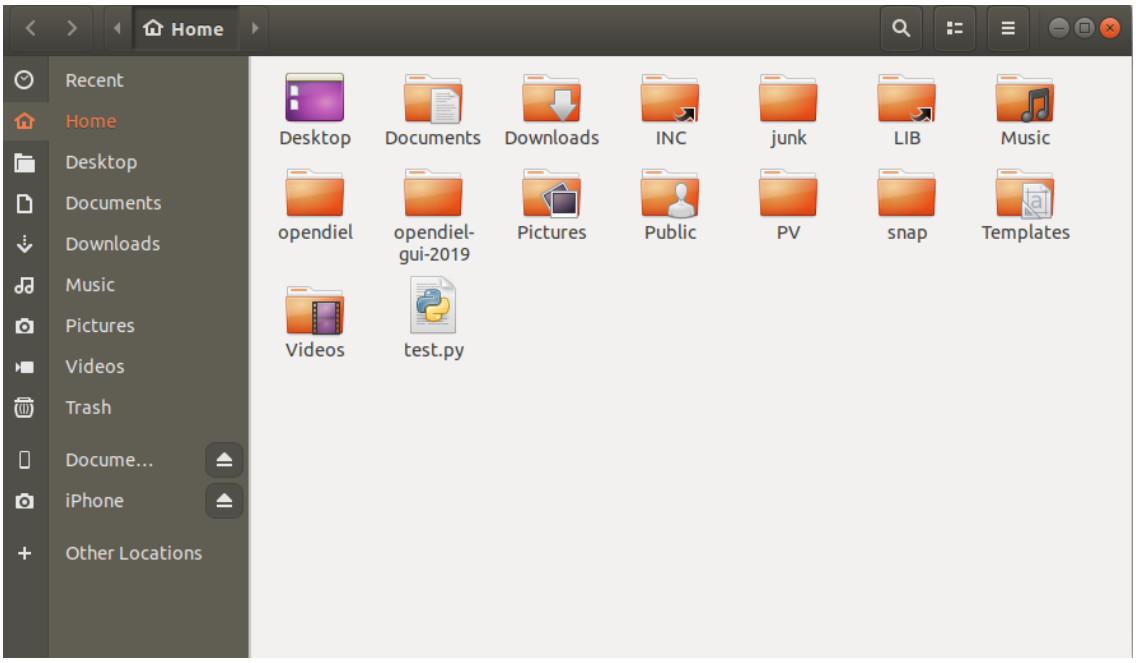
These steps can be extremely tedious. Running modules using openDIEL is not user friendly.

2) Method

2.1) GUI

2.1.1) What is a GUI

A GUI is a Graphical User Interface. A graphical user interface is primarily used to help with computer operations by providing visual indicators. Buttons and drop down menus are much easier to navigate through compared to the terminal. Even the file manager system on all operating systems is an example of a GUI.

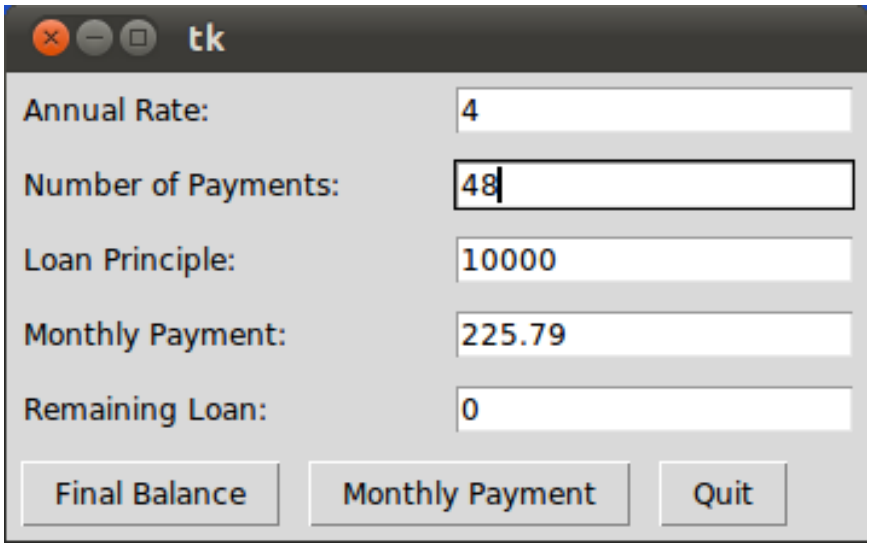


2.1.2) Use of the GUI

The GUI will be used in place of running commands inside the terminal. The GUI will help the user create modules and workflow as well as calculate the number of mpi processes.

2.1.3) How to create GUI

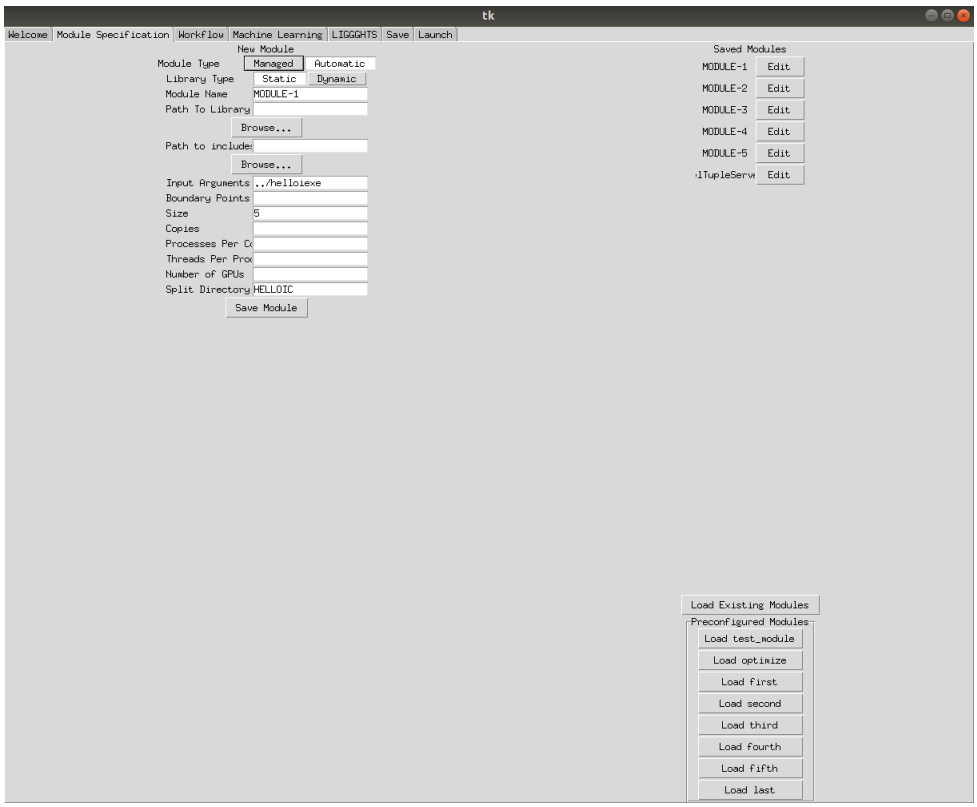
To create the GUI, a python GUI programming toolkit called Tkinter was used. Tkinter is included with python and the most commonly used python GUI programming toolkit. Tkinter is easy to learn and extremely accessible. Tkinter is also cross platform and very stable. However, the user must be familiar with python to utilize Tkinter. Tkinter is based around widgets and object oriented programming. Each widget is inherited from the widget class. Here is an example of widgets in Tkinter.



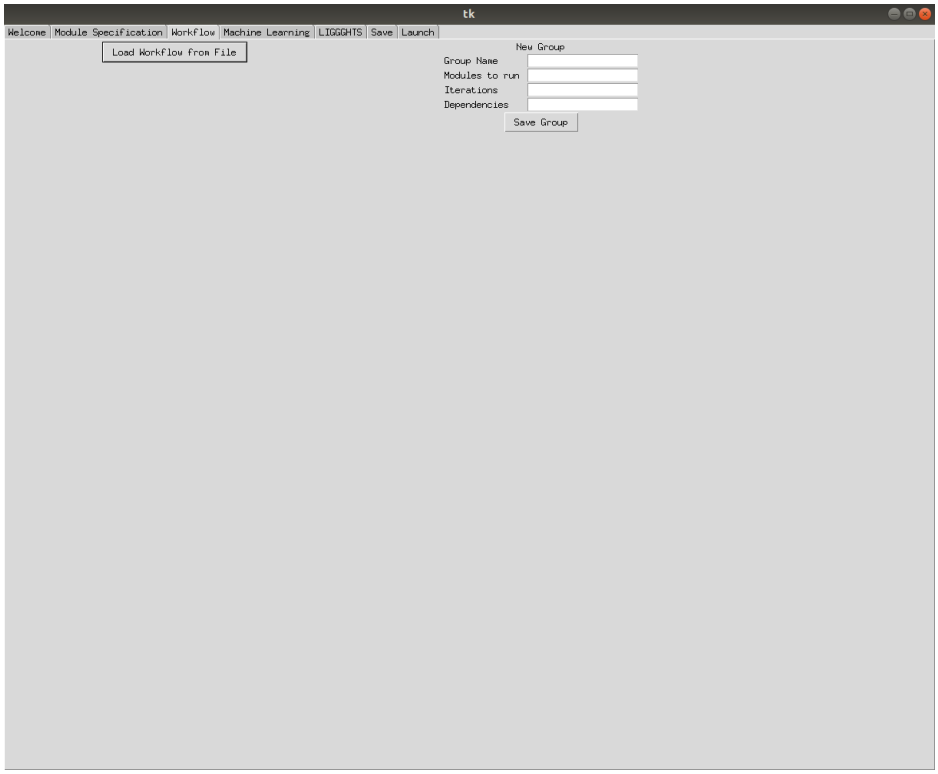
The user could use the entry box widgets to enter information and then proceed to save the information. In similar fashion, different widgets were used to help create module, workflow, and configuration files for the openDIEL GUI.

2.1.4) Functionality of GUI

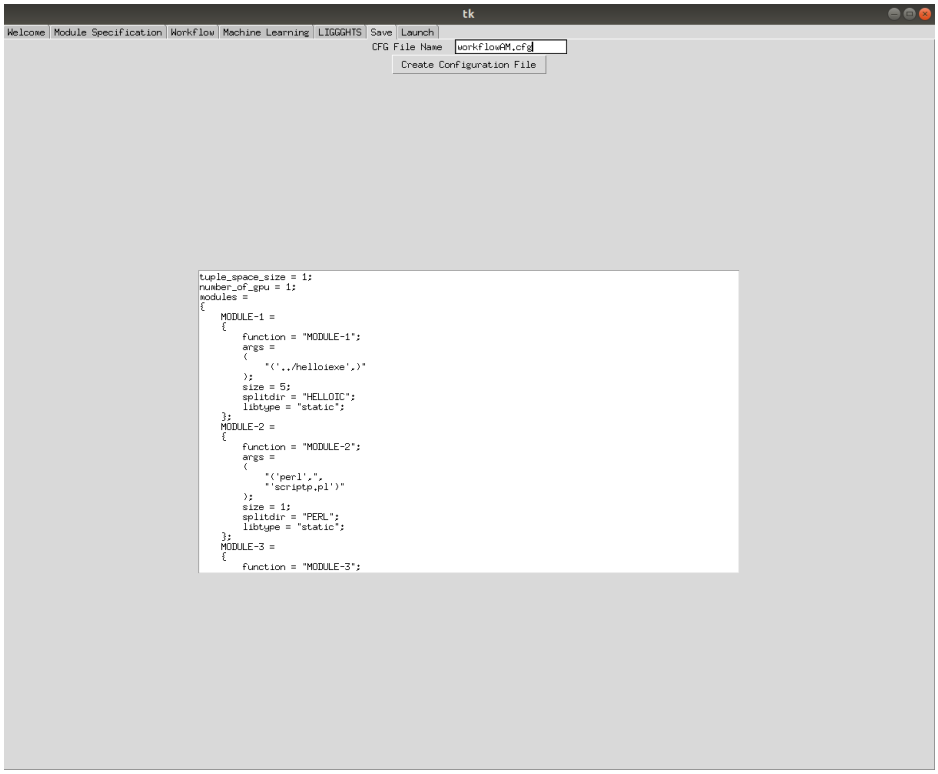
Step one in using the GUI is to create modules or load existing modules to be ran. Within the modules there a number of different attributes that the user must enter.



They will then proceed to the workflow section and add their created modules to groups. Once the user is done creating their groups they can add different dependencies as well as number of iterations.



Once the modules and workflow section has been created the user can save their work and use the button widget to create the configuration file.



After all these steps their modules are ready to be run. The user can launch their job and that is the end to the process. The mpirun command will be called and the number of mpi processes will be determined behind the scenes. This process using buttons and entry widgets makes creating jobs for openDIEL extremely simple.

2.1.5) Problems with current GUI (Tkinter)

However, Tkinter is not the perfect solution. Stylistically, Tkinter is not the most visually appealing and laying out widgets in an attractive manner can prove to be difficult. However, there was a solution to that problem.

2.2) Kivy

2.2.1) What is Kivy?

Kivy is an open source Python framework for rapid development of applications that make use of innovative user interface. Kivy can work across different kinds of platform such as Android, IOS, IpadOS, Windows, Linux, MacOS. Since its graphics engine is built over OpenGL ES 2, which uses a modern and fast graphics pipeline, it is GPU accelerated.

2.2.2) Why Kivy?

Kivy makes laying out widgets simple because it has its own design language. It also fits with mobile app development without needing to modify code. Moreover, kivy's toolkit comes with more than 20 widgets. All of them are highly extensible. Most of them are written in C using Cython and tested with a regression test.

2.2.3) How to install Kivy?

There are many ways to install Kivy. However, anaconda 3 is the recommended way. To begin with, install anaconda 3 from their website. https://www.anaconda.com/distribution/. There are two ways to install anaconda 3, one of them is through the command line and the other way is through the application. Both of them work fine, but installing through the command line is recommended.

After installing anaconda 3, there is a certain version of python that must be installed on the computer. Open the terminal and type "python —version". If the python version is 3.7 it means anaconda 3 was installed successfully.

Furthermore, in the command line type "conda install -c conda-forge kivy". It takes some time to fully install it. Then, enjoy your time with kivy.

2.2.4) How KIVY works?

In order to use Kivy, users need to become familiar with python. Then the user must learn how to use the kivy language. Kivy can also automatically format

widgets that are suitable for all platforms. However, Kivy needs a special language to define the layout which allows logic to be kept separate from the presentation. It is called the kv language. Kv language allows users to create widget trees in a declarative way. It binds widget properties to each other or to call back in a natural manner. Kv language is a language used to give the syntax of the kivy program a better view by representing all the elements in the program like classes, the other classes it is inheriting, widgets, and their properties and configurations. It is much more clear and understandable which means using the kv language makes your code much more clear and organized. However, it is not always used (if the program is of very few lines) but it is a good approach to understand how things work in kivy.

```
TabbedPanelItem:
    text: 'Module Specification'
    BoxLayout:
      orientation: 'horizontal'
      BoxLayout:
        orientation: 'vertical'
        Label:
          text: 'New Module'
        BoxLayout:
          orientation: 'horizontal'
          Label:
            text: 'Module Type'
            canvas.before:
              Color:
                rgba: 43/255, 101/255, 236/255, .5
              Rectangle:
                pos: self.pos
                size: self.size
          ToggleButton:
            id: manage
            text: 'Managed'
            group: 'Module_Type'
            background_normal: ''
            background_color: 43/255,101/255 , 236/255, .5
          ToggleButton:
            id: autom
            text: 'Automatic'
            group: 'Module_Type'
            background_normal: ''
            background_color: 43/255,101/255 , 236/255, .5
        BoxLayout:
          orientation: 'horizontal'
          Label:
            id: 'libtype'
            text: 'Library Type'
            canvas.before:
              Color:
                rgba: 43/255, 101/255, 236/255, 1
              Rectangle:
                pos: self.pos
                size: self.size
          ToggleButton:
            id: stat
            text: 'Static'
            group: 'Library_Type'
            background_normal: ''
            background_color: 43/255,101/255 , 236/255, 1
          ToggleButton:
            id: dyn
            text: 'Dynamic'
```

```
            group: 'Library_Type'
            background_normal: ''
            background_color: 43/255,101/255 , 236/255, 1
        BoxLayout:
            orientation: 'horizontal'
            Label:
                text: 'Module Name'
                canvas.before:
                    Color:
                        rgba: 43/255, 101/255, 236/255, .5
                    Rectangle:
                        pos: self.pos
                        size: self.size
            TextInput:
                id: func
                #text: ('<None>' if (not app.fc_filename or not root.show_details) else str(pathlib.Path(app.fc_filename).parent)) #path of doc
        BoxLayout:
            orientation: 'horizontal'
            Label:
                text: 'Path to Library'
                canvas.before:
                    Color:
                        rgba: 43/255, 101/255, 236/255, 1
                    Rectangle:
                        pos: self.pos
                        size: self.size
            TextInput:
                id : library
                #text: ('<None>' if (not app.fc_filename or not root.show_details) else str(pathlib.Path(app.fc_filename).parent)) #path of doc
        Button:
            text: 'Browse...'
            on_release: app.BrowsePath(check=0)
            background_normal: ''
            background_color: 43/255,101/255 , 236/255, .5

          .

          .

          .

          .
        Button:
            text: 'Save Module'
            on_release: app.moudletab.save_module()
            background_normal: ''
            background_color: 43/255,101/255 , 236/255, 1
        Button:
            text: 'Save as Module File'
            on_release: app.moudletab.save_module()
            on_release: app.saveMF()
            background_normal: ''
            background_color: 43/255,101/255 , 236/255, .5
    BoxLayout:
        orientation: 'vertical'
        Label:
            text: 'Saved Modules'
            canvas.before:
                Color:
                    rgba: 43/255, 101/255, 236/255, .5
                Rectangle:
                    pos: self.pos
                    size: self.size
            size_hint_y: .067
        ScrollView:
            size_hint_x: 1
            size_hint_y: 1
            canvas.before:
```

```
            Color:
                rgba: 43/255, 101/255, 236/255, 1
            Rectangle:
                pos: self.pos
                size: self.size

        GridLayout:
            do_scroll_x: False
            do_scroll_y: True
            id:container
            spacing:1.5
            cols: 1
            height: 3000
            size_hint_y:None
    Button:
        text: 'Load Existing Modules'
        size_hint_y: .0625
        background_normal: ''
        background_color: 43/255,101/255 , 236/255, .5
        on_release:
            root.show_details = False
            #hide1.visible = True
            app.BrowsePath(check=2)
```

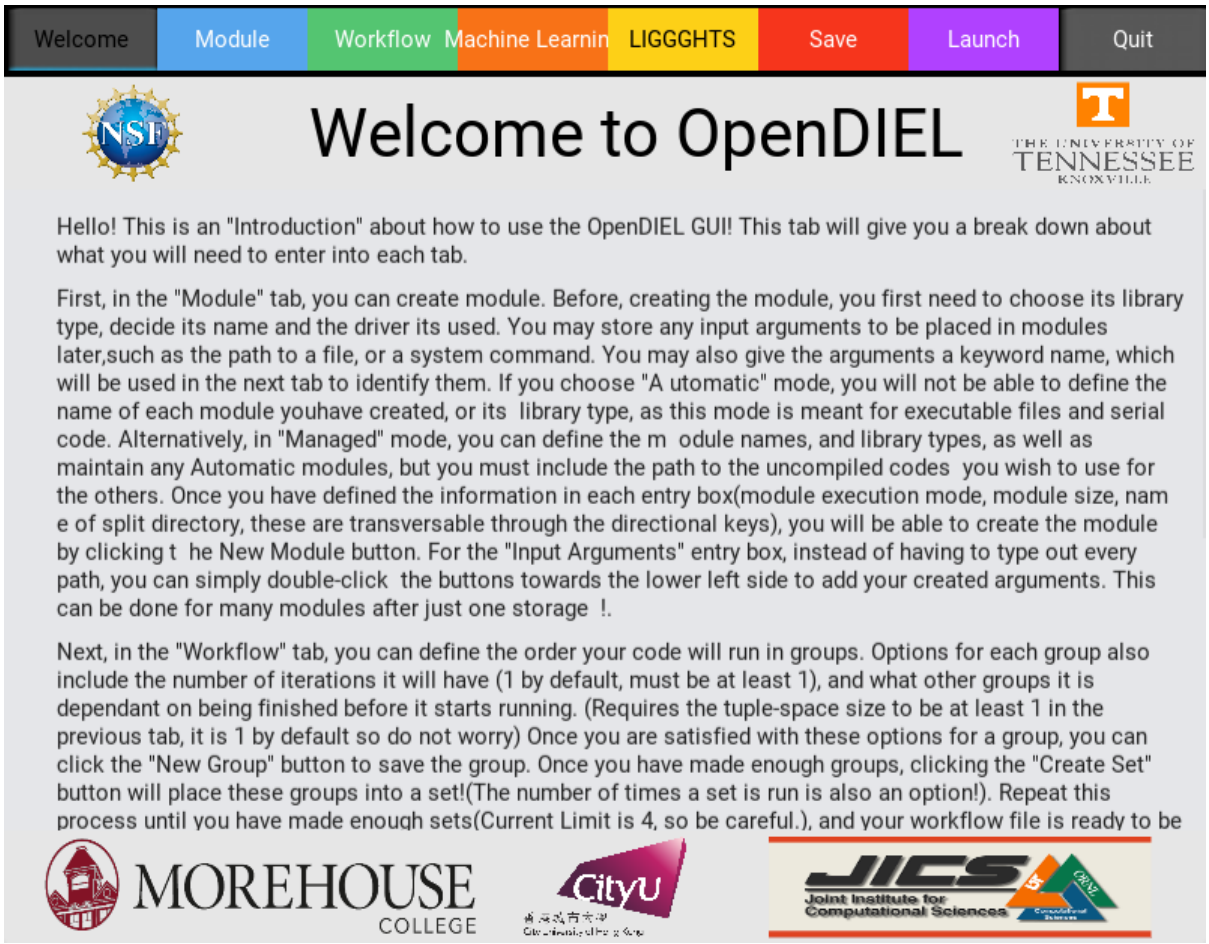
The above figures show the kivy code of the “module tab”. From the figure, it is shown that the page is divided into two parts. The left consists of several columns that show different kinds of details and data of the module. There are text input boxes, labels for the heading, buttons for browse and saving the changes of the module. On the right side, it will list all the modules of the configuration file after clicking the “ Load Existing Module” button

.

3) Result

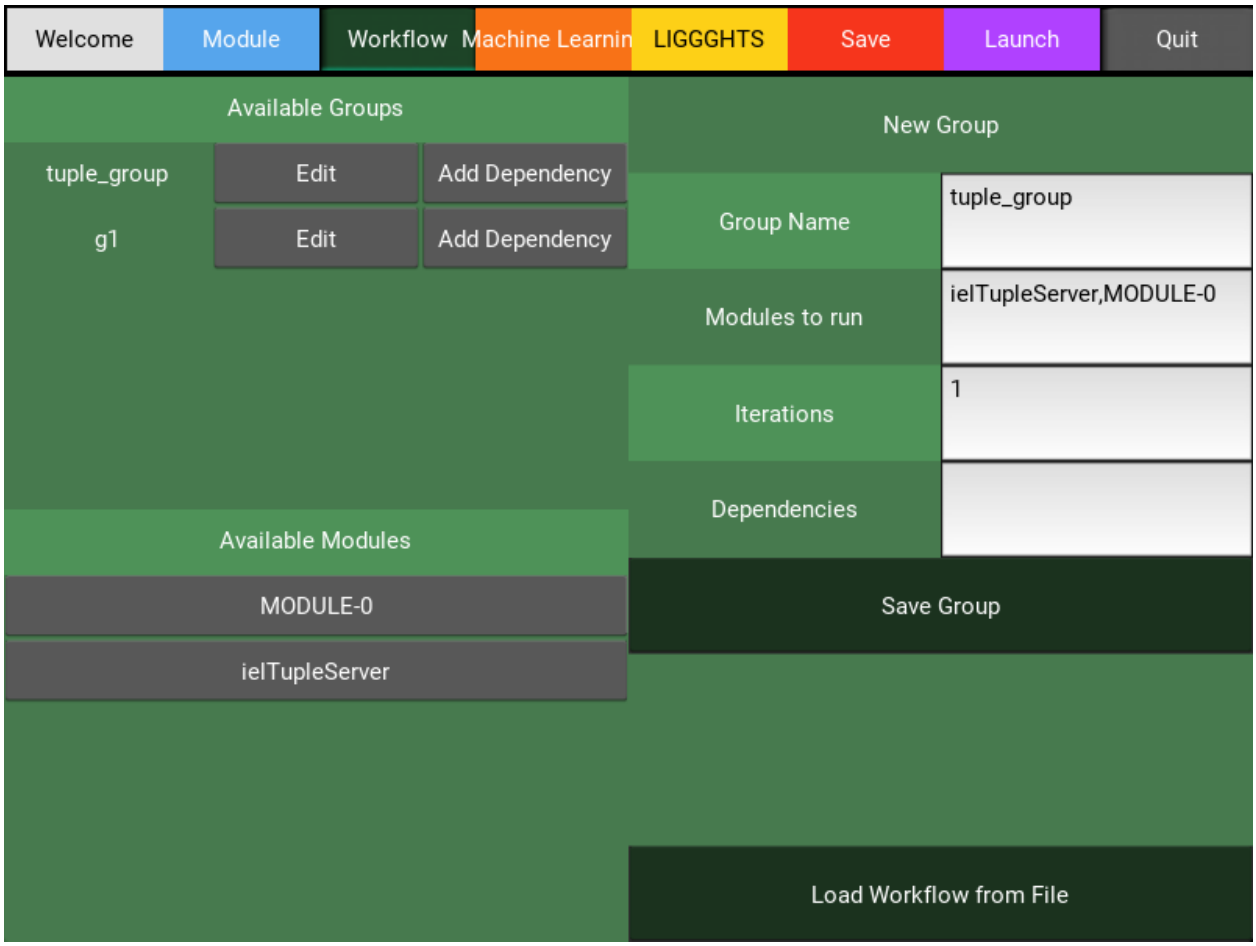
3.1) How we use Kivy?

The application has seven tabs which consist of many different applications. First, the “Welcome” tab, which is also the first tab the user sees once they run the application. In this tab, it will briefly described how the application works and how to use it.

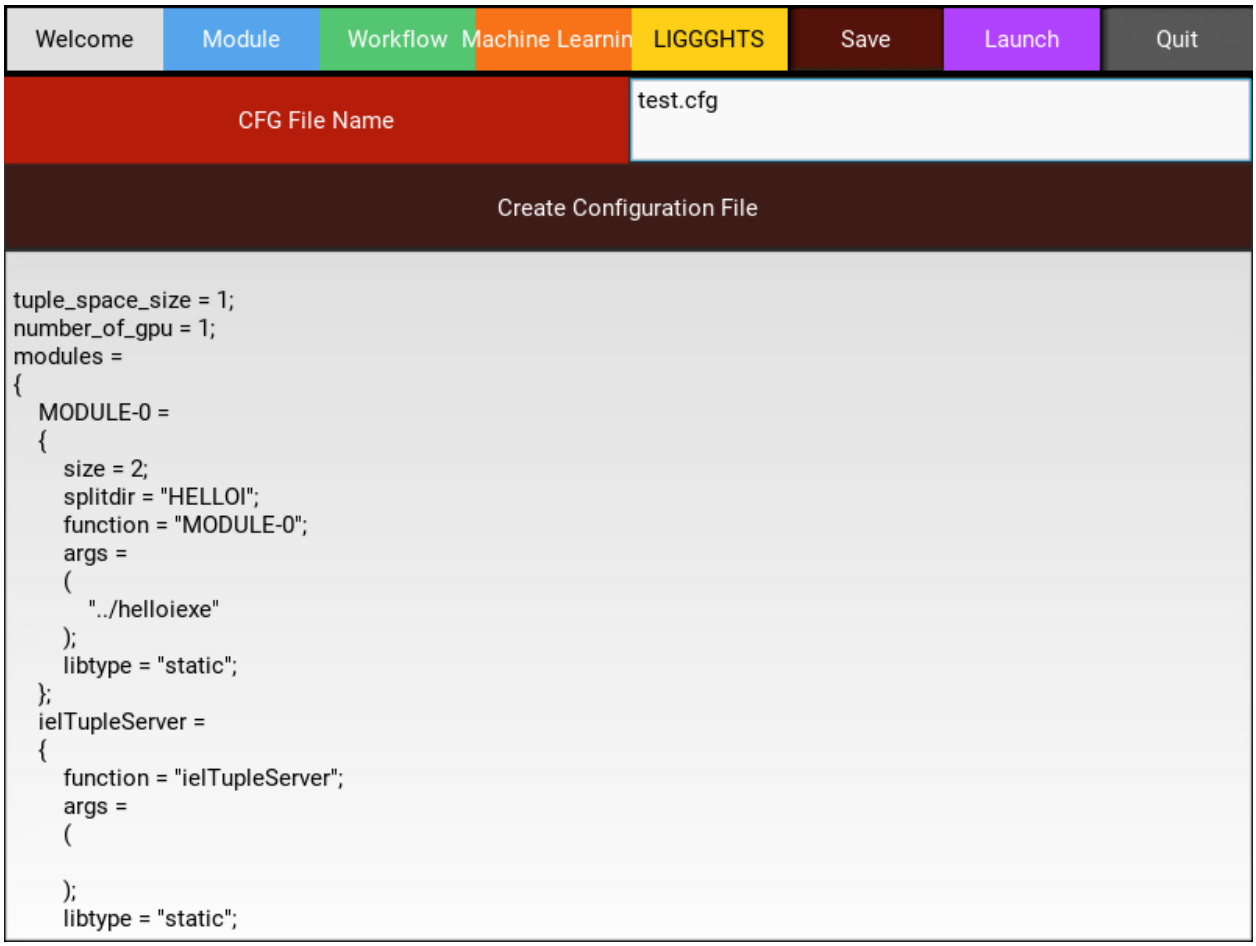


The second tab is the “Module” tab. In the module tab, new modules are created or old modules can be loaded. Before creating the module, users need to choose the library and module type. After, the user must enter the input arguments, size and split directory and of course the module name. After inserting the above data, save the module using the save module button. The saved modules will appear on the right side of the application which is under the “saved modules” section. If you choose “Automatic” mode, you can not define the name of the module that you created. In the “Managed” mode, users can define module name and library type, also you can decide the path. After loading the modules, users can make changes on it as well.

The third tab is the “Workflow” tab and defines the order that the groups run in. The number of iterations must be at least 1 and the user can set dependencies for groups. Save the groups and the configuration file can now be created. You can also load workflow from your own configuration file by clicking the “Load Workflow from File” button.

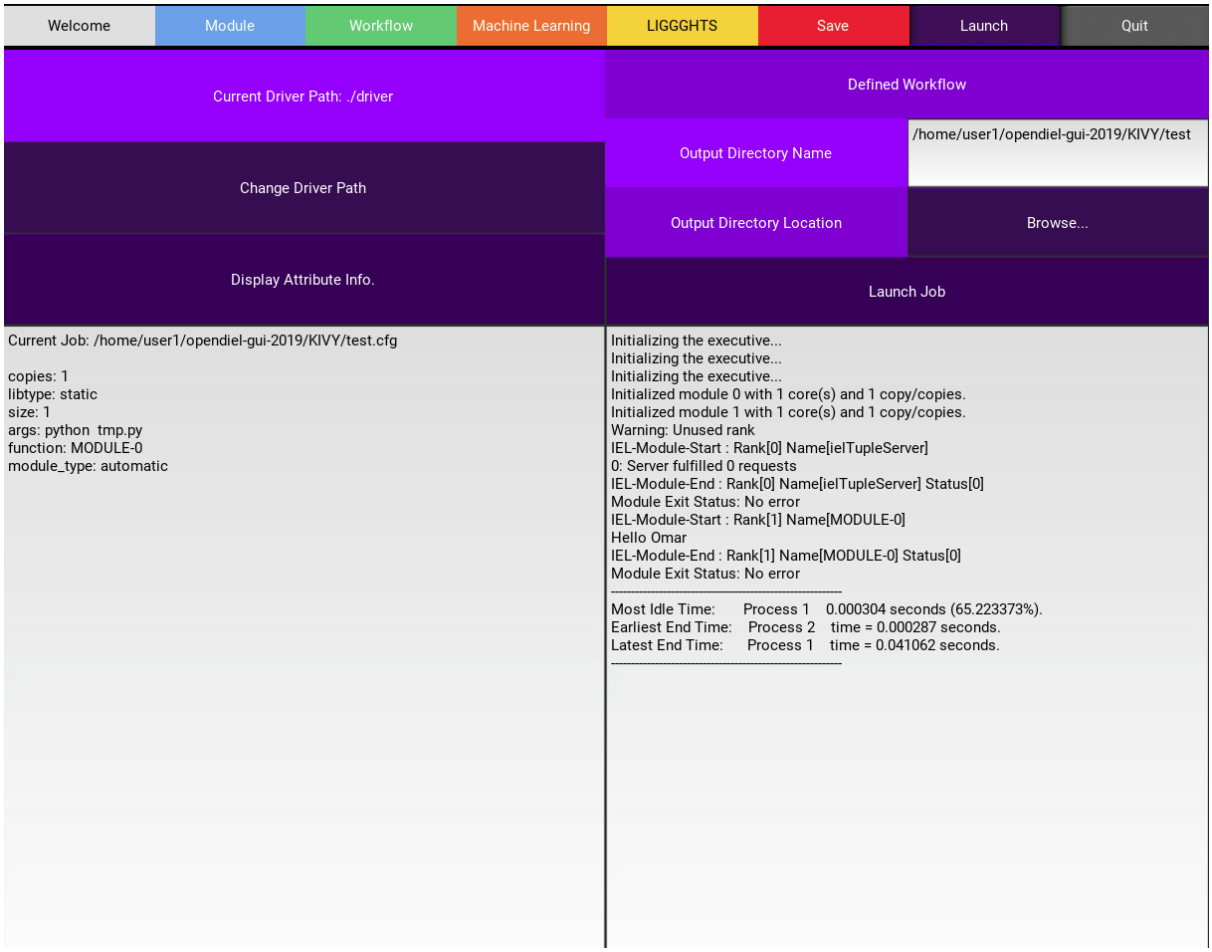


In the sixth tab, properly named the “Save” tab, the configuration file is created using the data in the previous tab and saved in the current directory.



Finally, users can go to the “Launch” tab and launch their openDIEL job. First, a driver needs to be chosen by clicking the “Change Driver Path”. Users can also use the dynamic one. The users need to choose the output directory location; the default directory will be the current directory. The user can now launch their job by clicking

the “Launch” job button and the result will be displayed in the text box.



4) Conclusion

Although many functionalities of the GUI work properly there are a few details that can be updated. For example, it would prove useful to allow users to access the openDIEL source code so that they could run the different examples. A new interface to the grid engine,

running liggghts on kivy, and adding more search methods to the grid engine such as Population Based Training with configuration file interface are just some things that can be improved in the GUI. There were some troubles in formatting the configuration files during the beginning of the project. The function for writing to the .cfg file needed some modification. In addition, more functionality was added to the GUI increase the ease of use. Minor details that made a big difference in usability. Creating a GUI to run modules on openDIEL was successful.