Creating a GUI to define workflows in openDIEL and adding support for GPU Deployment

By: Efosa Asemota, Frank Betancourt, Quindell Marshall
Mentor: Dr. Kwai Wong

Introduction

- openDIEL (Open Distributive Interoperable Executive Library) is a lightweight workflow engine designed for usage in HPC environments.
- Allows for the unification of many different modules into a single executable
- Written in C, and uses openMPI

Workflow files

- Workflow configuration files specify the workflow and what resources are to be used by each module
- First section contains modules: This is where you define what resources are needed by each module, such as the number of MPI processes, the number of GPUs, as well as defining where input and output needs to come from if needed.
- Second section contains the workflow: This is where the order of the modules is defined, and dependencies are specified, as well as the number of iterations that each group of modules needs to run

Intermodular Communication

There are two forms of communication:

- Direct, synchronous module-to-module communication with IEL functions IEL_put() and IEL_get()

- Indirect, asynchronous tuple space communication with IEL functions IEL_tput() and IEL_tget.

- Also supports using multiple tuple servers with a variety of different API calls for modules to communicate with tuple servers in different ways.

Driver

- As previously mentioned, all modules are unified under a single executable, referred to as the driver.
- Driver makes calls to MPI_Init(), MPI_Finalize(), and makes relevant calls to IEL member functions to read the configuration files, add modules, and start the entire IEL system

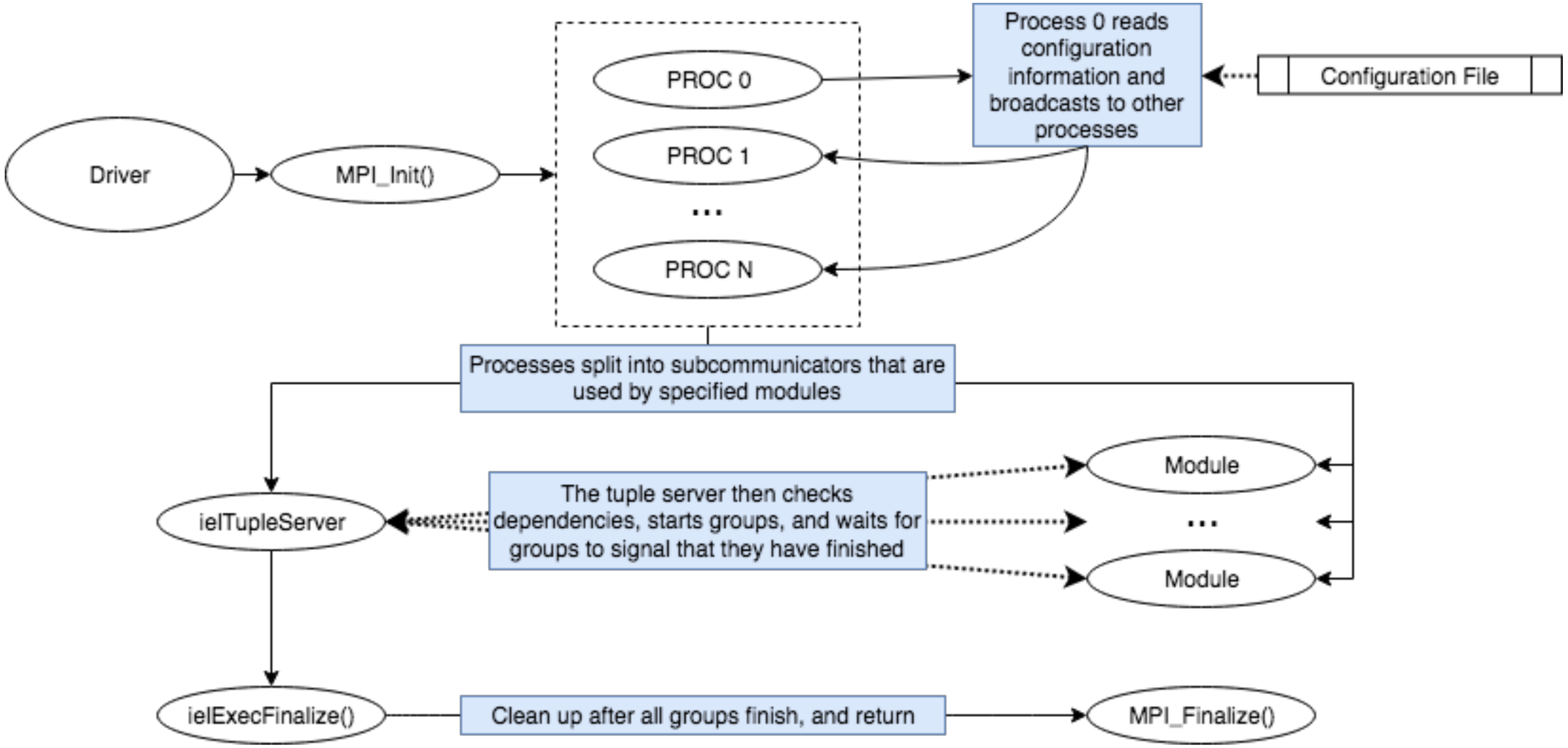

Driver
MPI_Init()
PROC 0
PROC 1
...
PROC N
Process 0 reads configuration information and broadcasts to other processes
Configuration File
Processes split into subcommunicators that are used by specified modules
ielTupleServer
The tuple server then checks dependencies, starts groups, and waits for groups to signal that they have finished
Module
...
Module
ielExecFinalize()
Clean up after all groups finish, and return
MPI_Finalize()

Figure 1: Simplified overview of MPI process usage

Managed and Automatic modules

There are two basic types of modules:

-Automatic: These are serial modules, with no calls to MPI functions, and they can simply be run in openDIEL with fork() and exec()

-Managed: These are parallel modules with calls to MPI functions. These require that any MPI_Init() or MPI_Finalize() calls are removed, since the driver will make these calls. Additionally, the MPI_COMM_WORLD created must be removed and replaced with the proper subcommunicator of the MPI_COMM_WORLD created by the driver.

Interface

-As previously mentioned, the interface is basically just a configuration file specified by the user

-Resources are specified on a per module basis through keywords, such as copies, processes_per_copy, num_gpu, threads_per_process, size, stdin, and splitdir

-Workflows are broken into two divisions: sets and groups

Application: GREP in parallel

-An application of openDIEL is parallel data searching

-Existing serial code can easily be parallelized under openDIEL by the following process:

1) Split data to be processed into smaller pieces
2) Run several unmodified copies of the script on different subsets of the data and direct the results of the script to files
3) Collect the output from the files and combine into a single file

Transit UDID search

-The need for such an application arose when existing Python scripts for a data search were not sufficiently fast for the data size.

-The goal was to be able to search through a dataset of 30 .csv files, totaling at about 180 million rows and 31 GB. Each row has 15 columns, and the goal is to search for and save all of the rows whose “UDID” column value matches the UDID value being searched for.

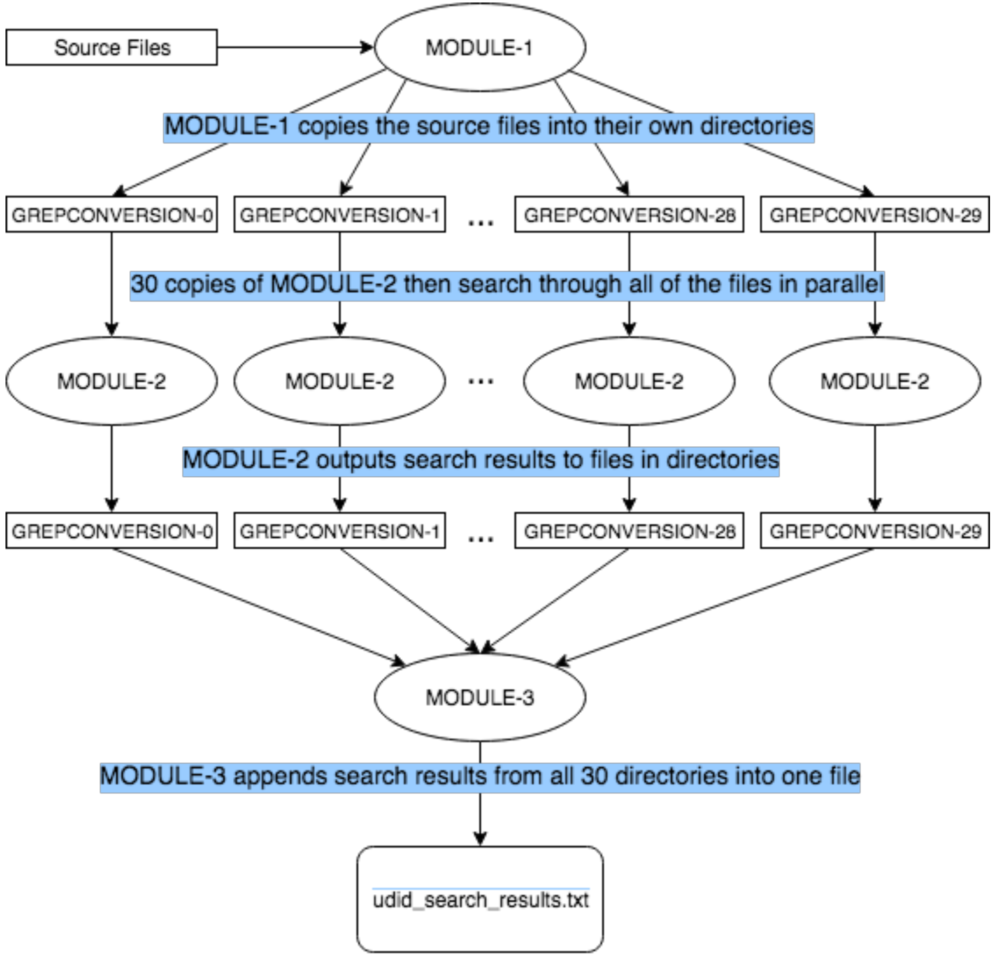
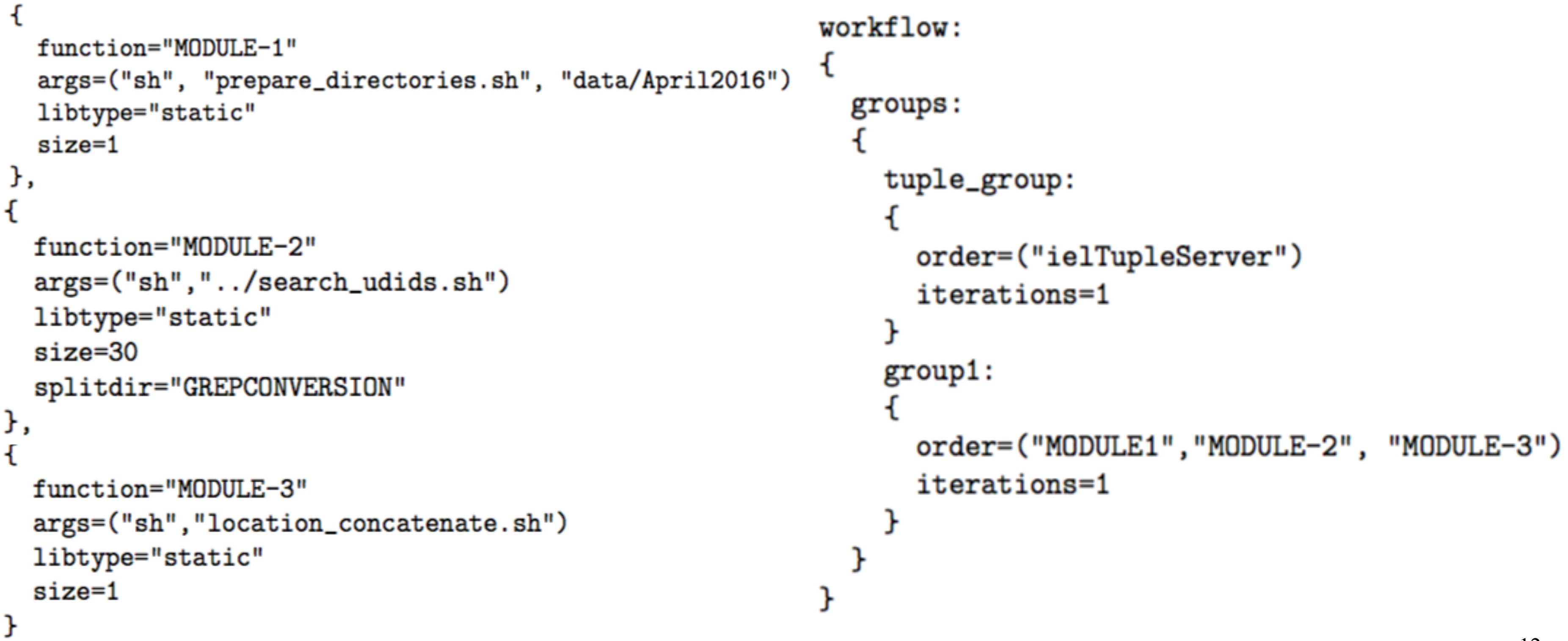

Source Files
MODULE-1
MODULE-1 copies the source files into their own directories
GREPCONVERSION-0
GREPCONVERSION-1
...
GREPCONVERSION-28
GREPCONVERSION-29
30 copies of MODULE-2 then search through all of the files in parallel
MODULE-2
MODULE-2
...
MODULE-2
MODULE-2
MODULE-2 outputs search results to files in directories
GREPCONVERSION-0
GREPCONVERSION-1
...
GREPCONVERSION-28
GREPCONVERSION-29
MODULE-3
MODULE-3 appends search results from all 30 directories into one file
udid_search_results.txt

Figure 2: Graph of parallel search with openDIEL

Workflow in openDIEL

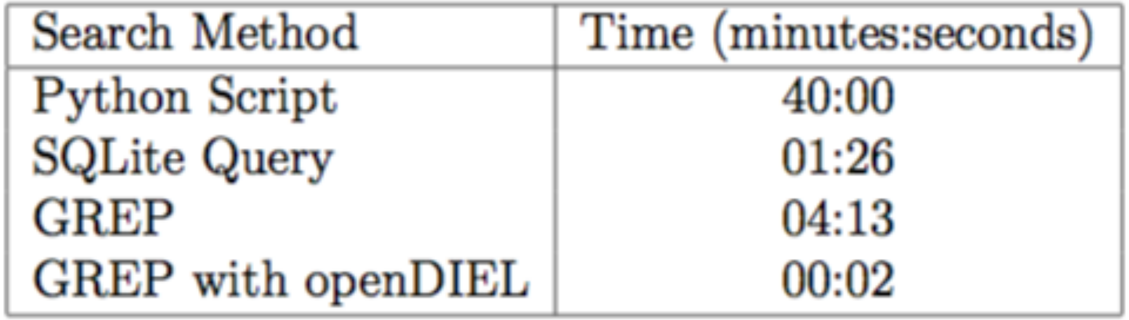


```
{
  function="MODULE-1"
  args=("sh", "prepare_directories.sh", "data/April2016")
  libtype="static"
  size=1
},
{
  function="MODULE-2"
  args=("sh","../search_udids.sh")
  libtype="static"
  size=30
  splitdir="GREPCONVERSION"
},
{
  function="MODULE-3"
  args=("sh","location_concatenate.sh")
  libtype="static"
  size=1
}
```

```
workflow:
{
  groups:
  {
    tuple_group:
    {
      order=("ielTupleServer")
      iterations=1
    }
    group1:
    {
      order=("MODULE1","MODULE-2", "MODULE-3")
      iterations=1
    }
  }
}
```

Figure 3: Parallel search workflow configuration file

Results



Search Method	Time (minutes:seconds)
Python Script	40:00
SQLite Query	01:26
GREP	04:13
GREP with openDIEL	00:02

Figure 4: Comparison of various methods to search for 1 UDID

Further improvements could likely be made by directing search results to an openDIEL tuple server, rather than conducting file I/O for saving and collecting search results.

Why do we need a GUI?

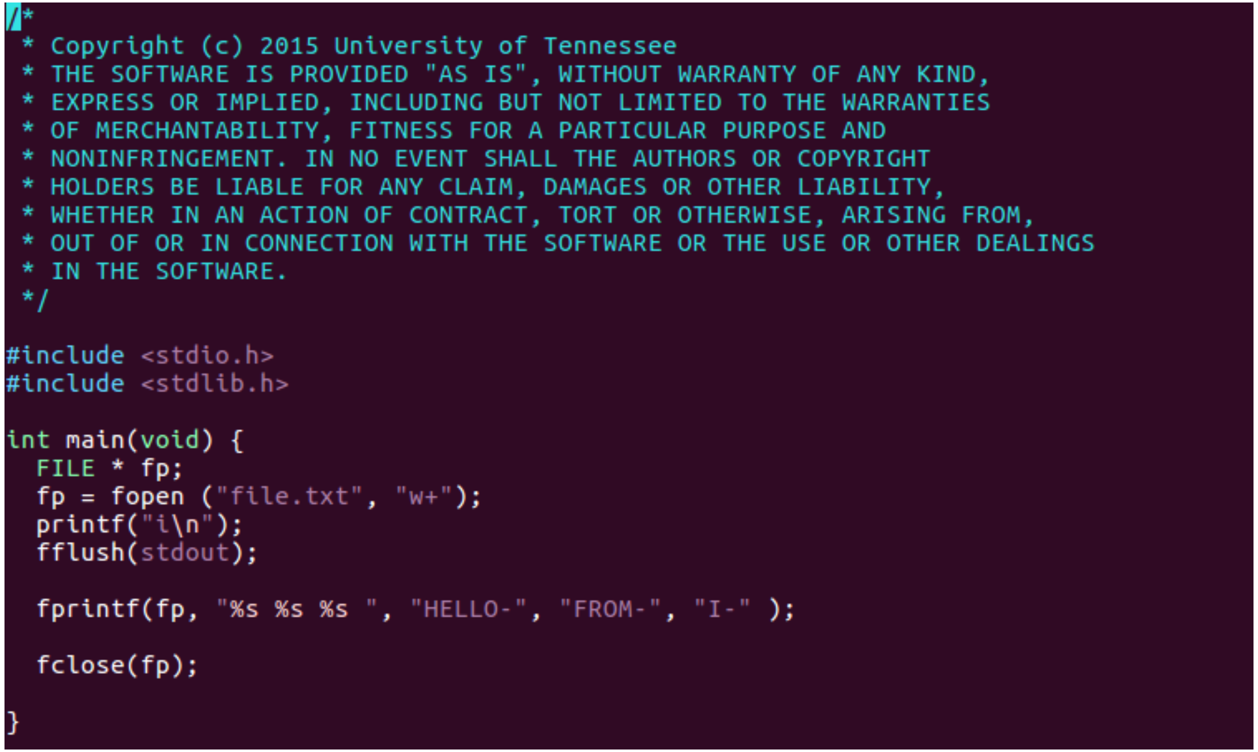
- The purpose of the GUI is to provide a much more user-friendly interface in order to execute openDIEL.

- Currently, in order for a user to execute openDIEL, they would have to go through a lot of tedious and time consuming tasks.

Why do we need a GUI? (contd.)

- One of the first steps that the user would have to do is to first convert their source code into a module/function by using the Python program ModMaker.py.
- The reason for this step is because in order for openDIEL to be executed, the user must convert their code(s) into a function.
- The user would need to repeat this step for as many modules that they would want to create.

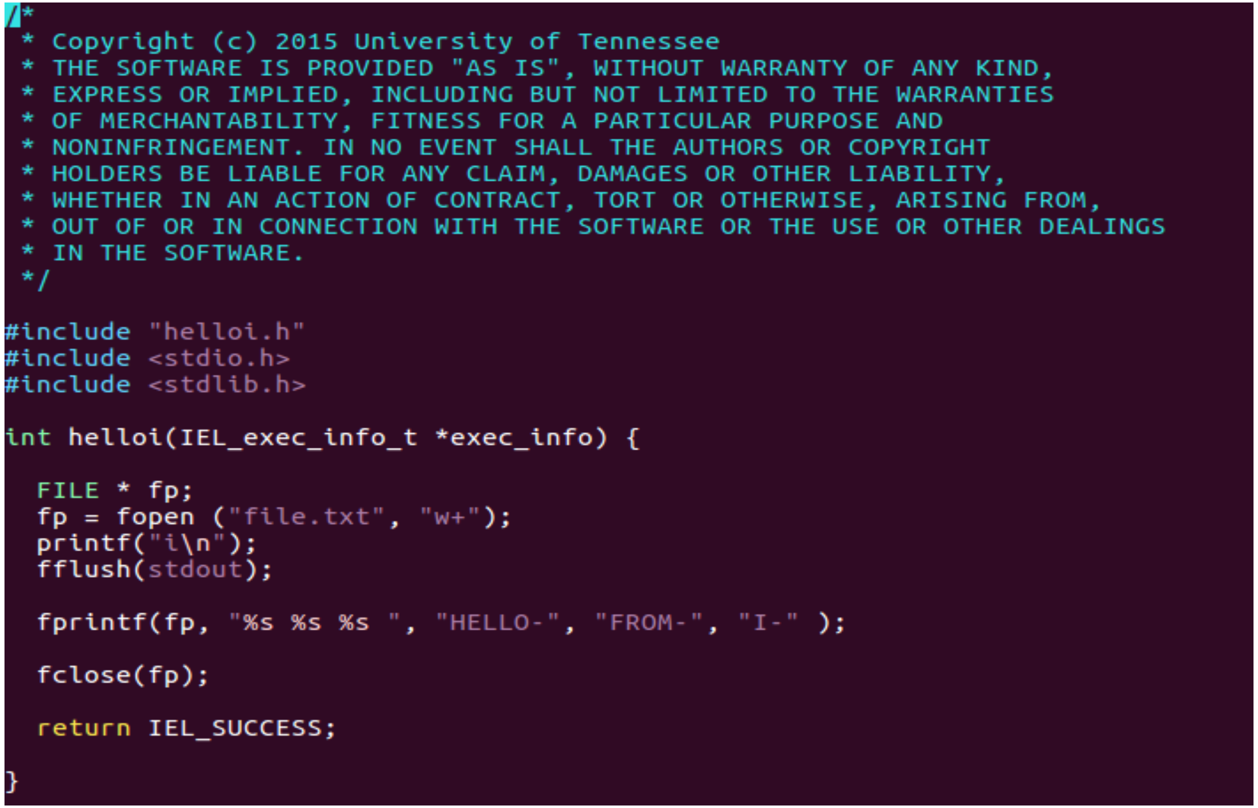
Original Code



```
/*
 * Copyright (c) 2015 University of Tennessee
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
 * IN THE SOFTWARE.
 */

#include <stdio.h>
#include <stdlib.h>

int main(void) {
  FILE * fp;
  fp = fopen ("file.txt", "w+");
  printf("i\n");
  fflush(stdout);

  fprintf(fp, "%s %s %s ", "HELLO-", "FROM-", "I-" );

  fclose(fp);

}
```

Figure 5: This figure is a simple example of a Hello World program

Converted Code from ModMaker.py

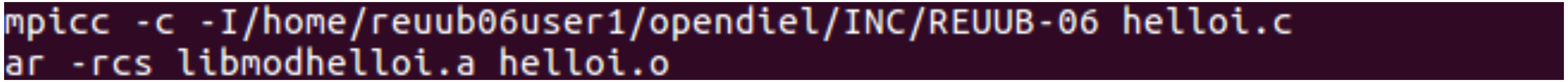


```
/*
 * Copyright (c) 2015 University of Tennessee
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
 * IN THE SOFTWARE.
 */

#include "helloi.h"
#include <stdio.h>
#include <stdlib.h>

int helloi(IEL_exec_info_t *exec_info) {

  FILE * fp;
  fp = fopen ("file.txt", "w+");
  printf("i\n");
  fflush(stdout);

  fprintf(fp, "%s %s %s ", "HELLO-", "FROM-", "I-" );

  fclose(fp);

  return IEL_SUCCESS;

}
```

Figure 6: This is the result of the Hello World program after it has been converted by ModMaker.py

Why do we need a GUI? (contd.)

- The next step would be for the user to also compile each module as library.
- In order to do that, the user would need to enter the following commands as shown in the following figure.
- Again, the user would need to repeat this step for as many modules that they would want to create.



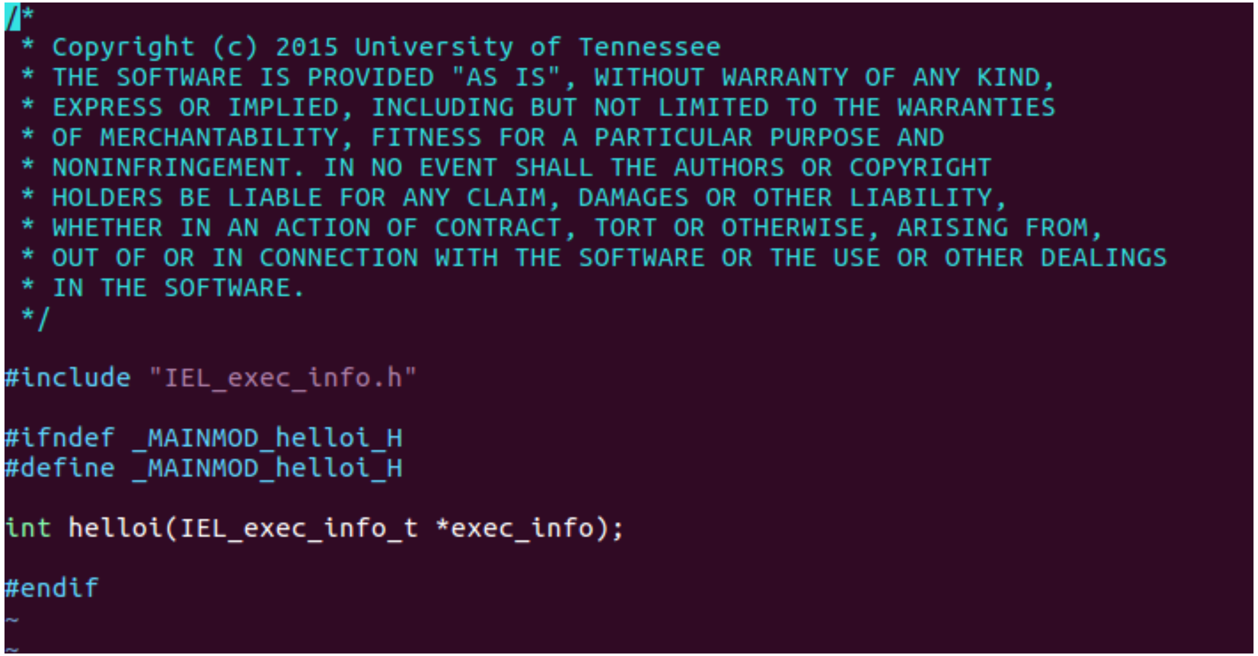
```
mpicc -c -I/home/reuub06user1/opendiel/INC/REUUB-06 helloi.c
ar -rcs libmodhelloi.a helloi.o
```

Figure 7: Commands for compiling a module as a library.

Why do we need a GUI? (contd.)

- The next step would be for the user to create a header file for each module that can be included into the Driver.c code.
- This step is very simple as all the user would need to do is follow this layout as shown in the following figure.
- The user again would repeat this step for as many modules that they would want to create.

Header File Creation

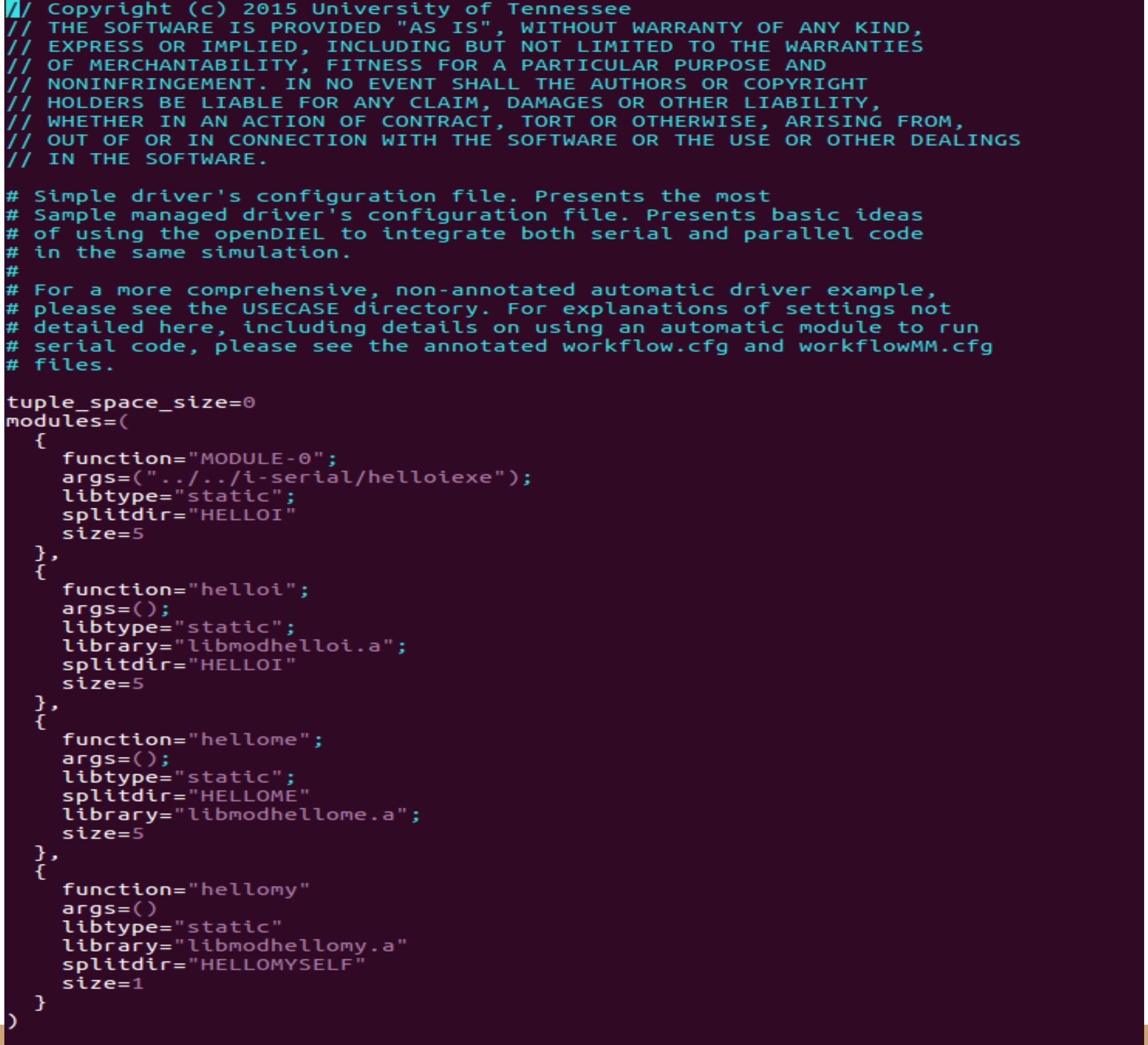


```
/*
 * Copyright (c) 2015 University of Tennessee
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
 * IN THE SOFTWARE.
 */

#include "IEL_exec_info.h"

#ifndef _MAINMOD_helloi_H
#define _MAINMOD_helloi_H

int helloi(IEL_exec_info_t *exec_info);

#endif
~
~
```

Figure 8: Example of a Header File for a module.

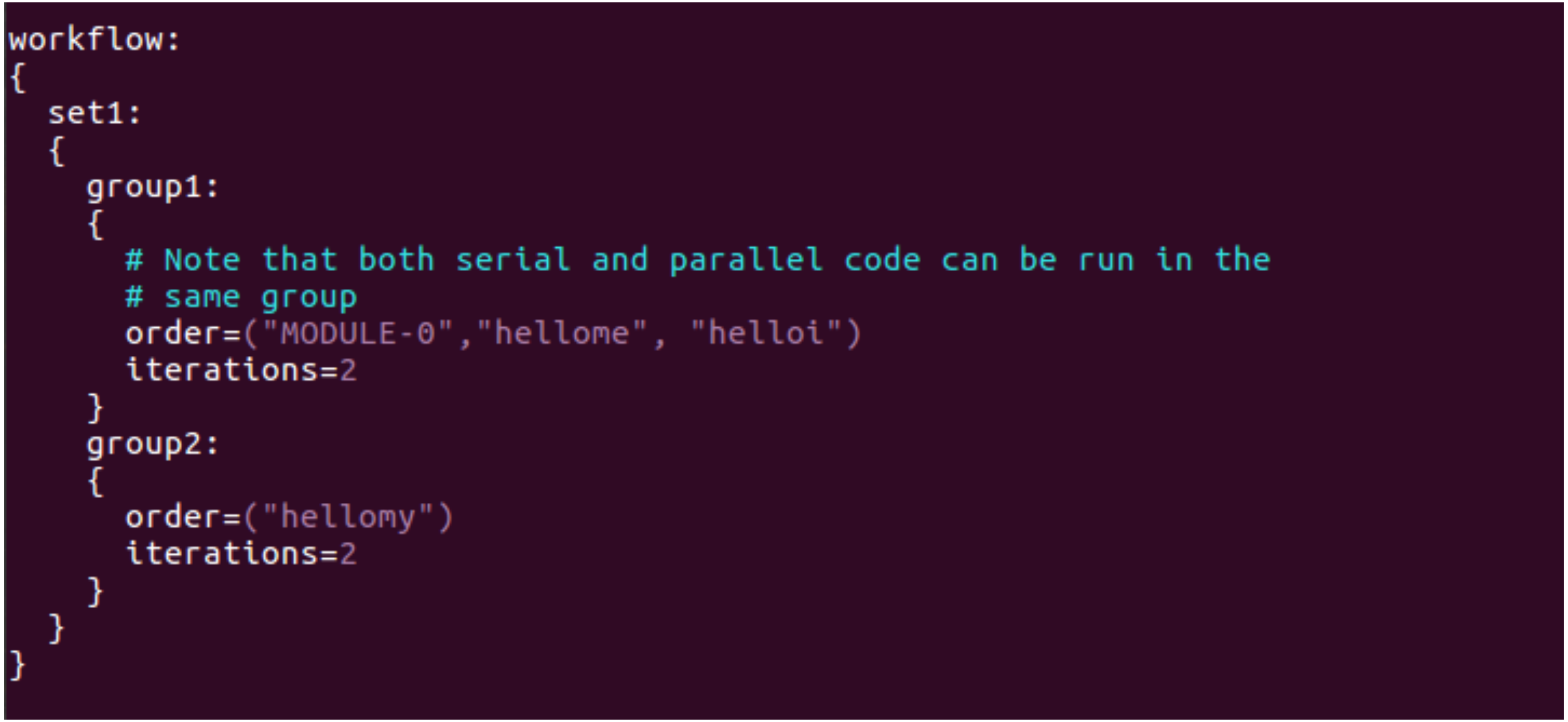
Why do we need a GUI? (contd.)

- The next step would be for the user to create a workflow configuration file.
- The workflow configuration file is divided into two sections: the module section and the workflow section.
- The purpose of the module section of the configuration file is to define the existence of each module.
- The purpose of the workflow section is to define how each module will run.
- The following figures illustrate the format of a typical configuration file.



```
// Copyright (c) 2015 University of Tennessee
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
// EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
// OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
// HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
// WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
// IN THE SOFTWARE.

# Simple driver's configuration file. Presents the most
# Sample managed driver's configuration file. Presents basic ideas
# of using the openDIEL to integrate both serial and parallel code
# in the same simulation.
#
# For a more comprehensive, non-annotated automatic driver example,
# please see the USECASE directory. For explanations of settings not
# detailed here, including details on using an automatic module to run
# serial code, please see the annotated workflow.cfg and workflowMM.cfg
# files.

tuple_space_size=0
modules=(
  {
    function="MODULE-0";
    args=("../../i-serial/helloiexe");
    libtype="static";
    splitdir="HELLOI"
    size=5
  },
  {
    function="helloi";
    args=();
    libtype="static";
    library="libmodhelloi.a";
    splitdir="HELLOI"
    size=5
  },
  {
    function="hellome";
    args=();
    libtype="static";
    splitdir="HELLOME"
    library="libmodhellome.a";
    size=5
  },
  {
    function="hellomy"
    args=()
    libtype="static"
    library="libmodhellomy.a"
    splitdir="HELLOMYSELF"
    size=1
  }
)
```

Figure 9: Module Section of the Configuration File.

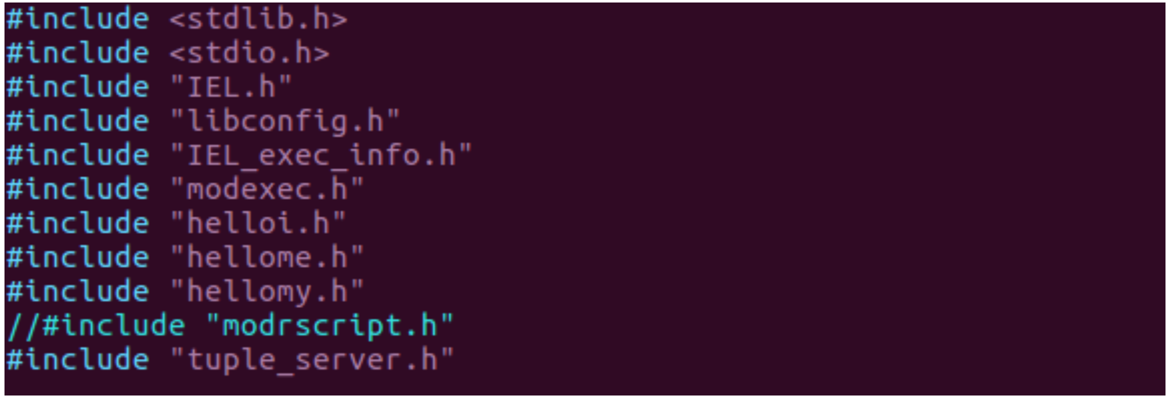


```
workflow:
{
  set1:
  {
    group1:
    {
      # Note that both serial and parallel code can be run in the
      # same group
      order=("MODULE-0","hellome", "helloi")
      iterations=2
    }
    group2:
    {
      order=("hellomy")
      iterations=2
    }
  }
}
```

Figure 10: The workflow section of the Configuration File

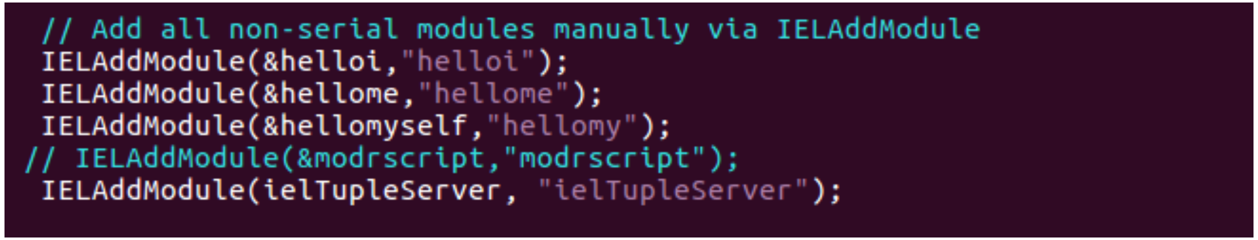
Why do we need a GUI? (contd.)

- The next step would be for the user to edit the Driver.c code by including each of the header files that correspond to each module they created.
- They would also go down in the code to the IELAddModule function call and pass as arguments a function pointer to the module and the name of the module as a string argument.
- The following figures detail these steps.



```
#include <stdlib.h>
#include <stdio.h>
#include "IEL.h"
#include "libconfig.h"
#include "IEL_exec_info.h"
#include "modexec.h"
#include "helloi.h"
#include "hellome.h"
#include "hellomy.h"
//#include "modrscript.h"
#include "tuple_server.h"
```

Figure 11: Module Header Files included in the Driver.c code.



```
  // Add all non-serial modules manually via IELAddModule
  IELAddModule(&helloi,"helloi");
  IELAddModule(&hellome,"hellome");
  IELAddModule(&hellomyself,"hellomy");
 // IELAddModule(&modrscript,"modrscript");
  IELAddModule(ielTupleServer, "ielTupleServer");
```

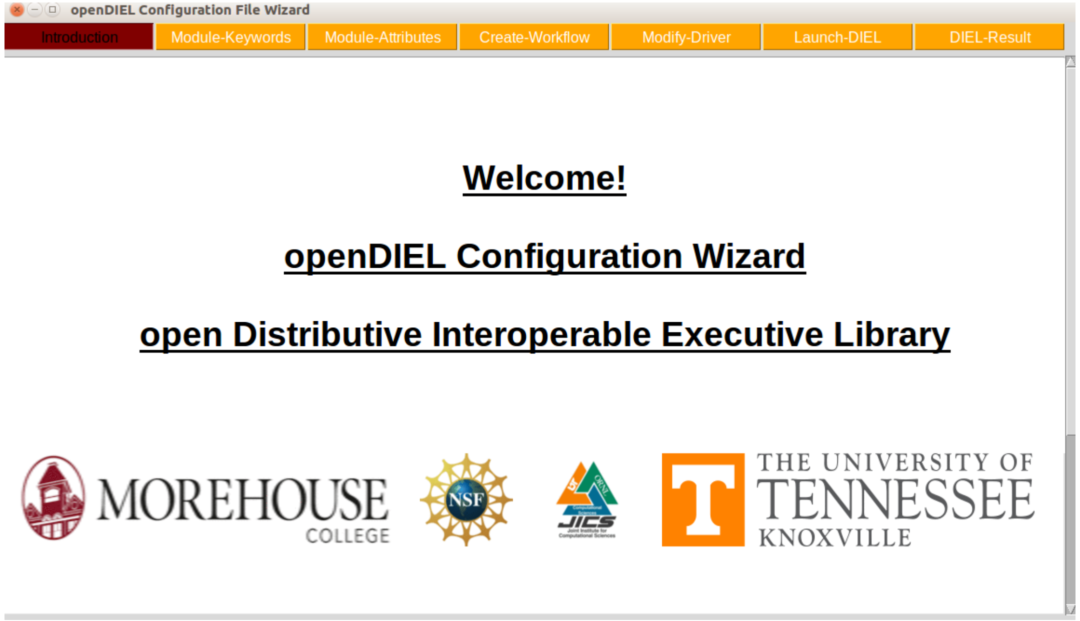
Figure 12: IELAddModule function calls for each module.

Why do we need a GUI? (contd.)

- Next, the user would need to link all of the compiled libraries to the Driver.c code and compile the Driver.
- Lastly, the user would need to run the Driver executable with the configuration file.

- With the assistance of a Graphical User Interface, the trouble of trying to do each step would be mitigated, as the GUI will seek to handle most of the responsibility for the user and to also speed up the process of preparing and running modules through openDIEL.

How the GUI Solves These Issues!



Welcome!

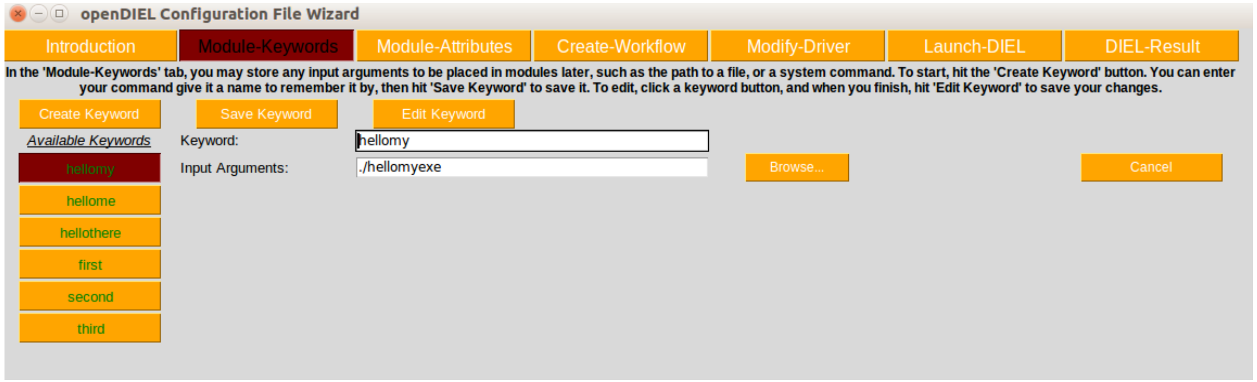
openDIEL Configuration Wizard

open Distributive Interoperable Executive Library

Figure 13: Introduction Tab

The openDIEL Graphical User Interface, constructed in Python, provides a much simpler, faster, well-laid-out alternative to the module construction process. Be it inside the *Introduction* tab or another section, there are short instructions to guide the user on what they need to do.

Keyword and Module Tabs

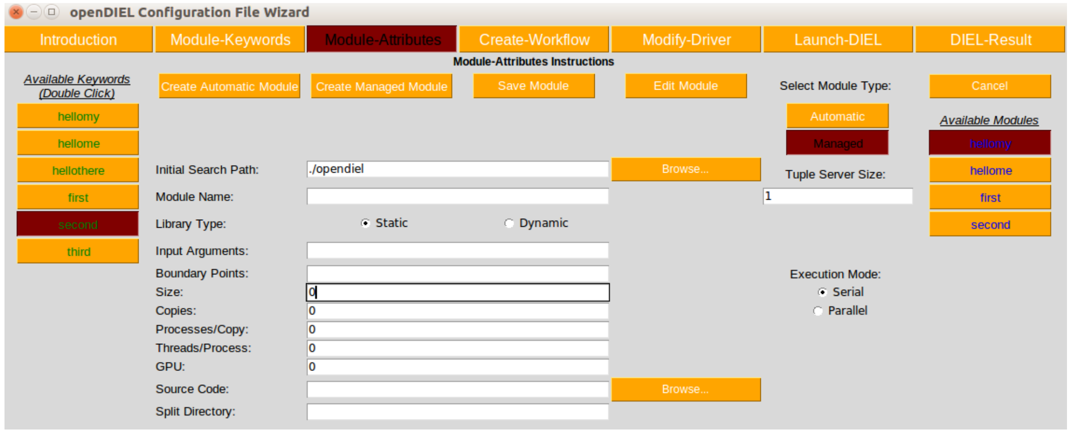

openDIEL Configuration File Wizard

In the 'Module-Keywords' tab, you may store any input arguments to be placed in modules later, such as the path to a file, or a system command. To start, hit the 'Create Keyword' button. You can enter your command give it a name to remember it by, then hit 'Save Keyword' to save it. To edit, click a keyword button, and when you finish, hit 'Edit Keyword' to save your changes.

Keyword: hellomy

Input Arguments: ./hellomyexe

Figure 14: Module Keywords Tab


openDIEL Configuration File Wizard

Select Module Type: Managed

Initial Search Path: ./opendiel

Module Name:

Library Type: [✓ Static] Dynamic

Input Arguments:

Boundary Points:

Size: 0

Copies: 0

Processes/Copy: 0

Threads/Process: 0

GPU: 0

Source Code:

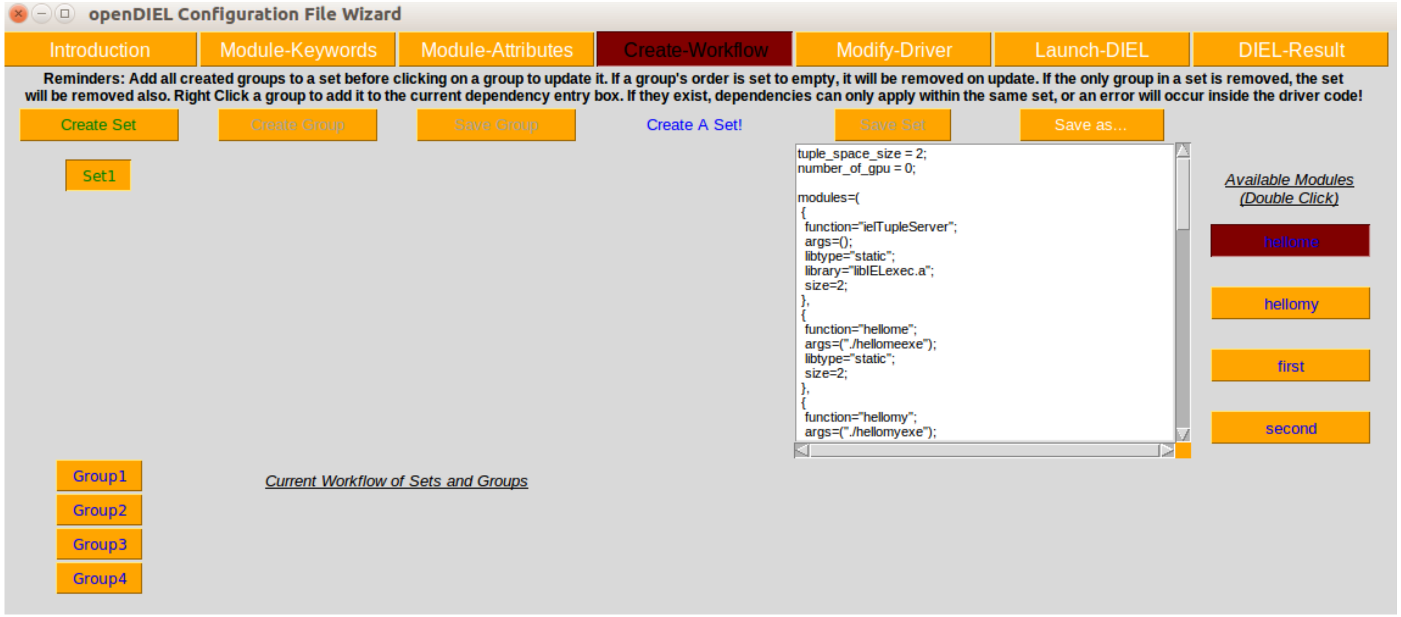
Split Directory:

Tuple Server Size: 1

Execution Mode: [✓ Serial] Parallel

Figure 15: Module Attributes Tab

Workflow Tab

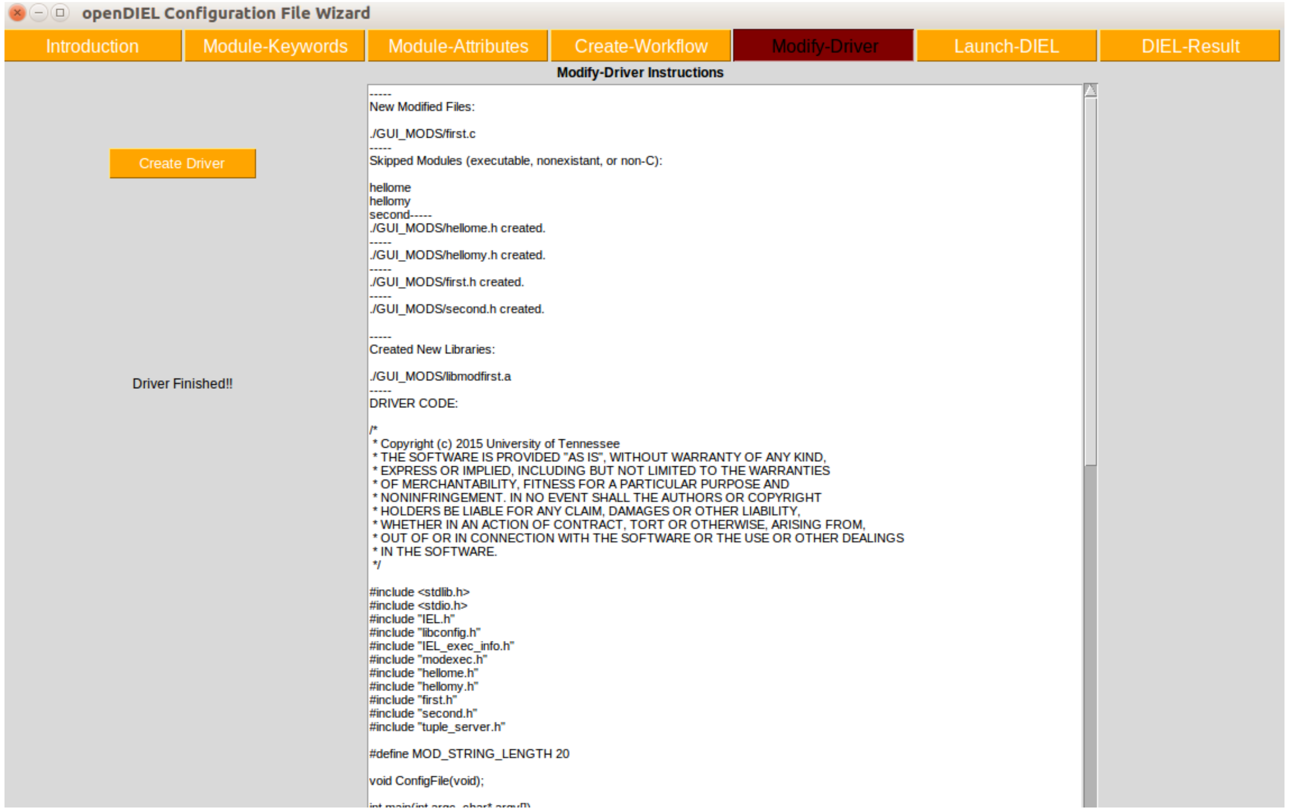


Reminders: Add all created groups to a set before clicking on a group to update it. If a group's order is set to empty, it will be removed on update. If the only group in a set is removed, the set will be removed also. Right Click a group to add it to the current dependency entry box. If they exist, dependencies can only apply within the same set, or an error will occur inside the driver code!

```
tuple_space_size = 2;
number_of_gpu = 0;

modules=(
 {
  function="ielTupleServer";
  args=();
  libtype="static";
  library="libIELexec.a";
  size=2;
 },
 {
  function="hellome";
  args=("./hellomeexe");
  libtype="static";
  size=2;
 },
 {
  function="hellomy";
  args=("./hellomyexe");
```

Figure 16: Create Workflow Tab

Driver Tab



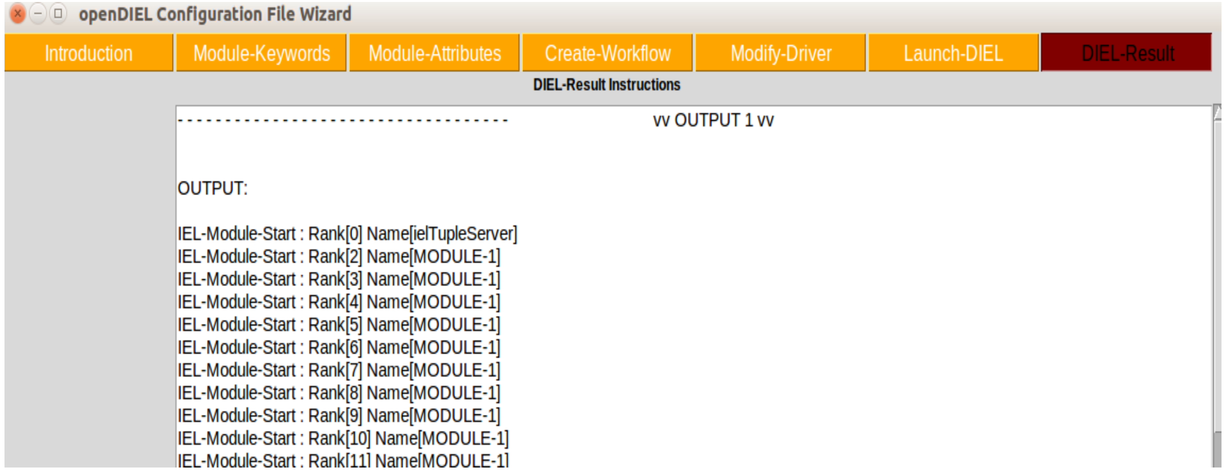
Modify-Driver Instructions

Driver Finished!!

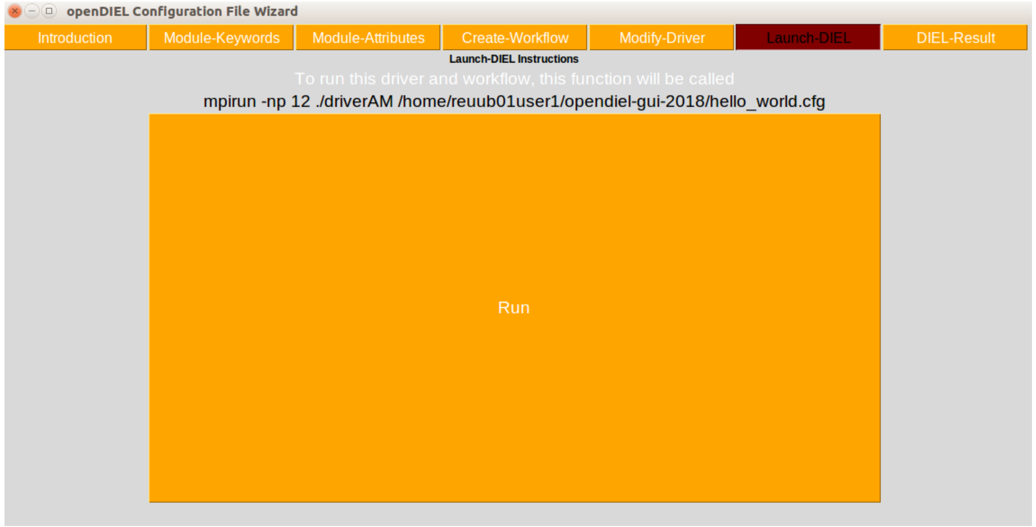
```
-----
New Modified Files:

./GUI_MODS/first.c
-----
Skipped Modules (executable, nonexistant, or non-C):

hellome
hellomy
second-----
./GUI_MODS/hellome.h created.
-----
./GUI_MODS/hellomy.h created.
-----
./GUI_MODS/first.h created.
-----
./GUI_MODS/second.h created.

-----
Created New Libraries:

./GUI_MODS/libmodfirst.a
-----
DRIVER CODE:

/*
 * Copyright (c) 2015 University of Tennessee
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
 * IN THE SOFTWARE.
 */

#include <stdlib.h>
#include <stdio.h>
#include "IEL.h"
#include "libconfig.h"
#include "IEL_exec_info.h"
#include "modexec.h"
#include "hellome.h"
#include "hellomy.h"
#include "first.h"
#include "second.h"
#include "tuple_server.h"

#define MOD_STRING_LENGTH 20

void ConfigFile(void);
```

Figure 17: Modify Driver Tab

Launch and Output Tabs



```
----------------------------------          vv OUTPUT 1 vv


OUTPUT:

IEL-Module-Start : Rank[0] Name[ielTupleServer]
IEL-Module-Start : Rank[2] Name[MODULE-1]
IEL-Module-Start : Rank[3] Name[MODULE-1]
IEL-Module-Start : Rank[4] Name[MODULE-1]
IEL-Module-Start : Rank[5] Name[MODULE-1]
IEL-Module-Start : Rank[6] Name[MODULE-1]
IEL-Module-Start : Rank[7] Name[MODULE-1]
IEL-Module-Start : Rank[8] Name[MODULE-1]
IEL-Module-Start : Rank[9] Name[MODULE-1]
IEL-Module-Start : Rank[10] Name[MODULE-1]
IEL-Module-Start : Rank[11] Name[MODULE-1]
```



To run this driver and workflow, this function will be called

```
mpirun -np 12 ./driverAM /home/reuub01user1/opendiel-gui-2018/hello_world.cfg
```

Figures 18 and 19: Launch-DIEL and DIEL-Result Tabs

Any

Questions?