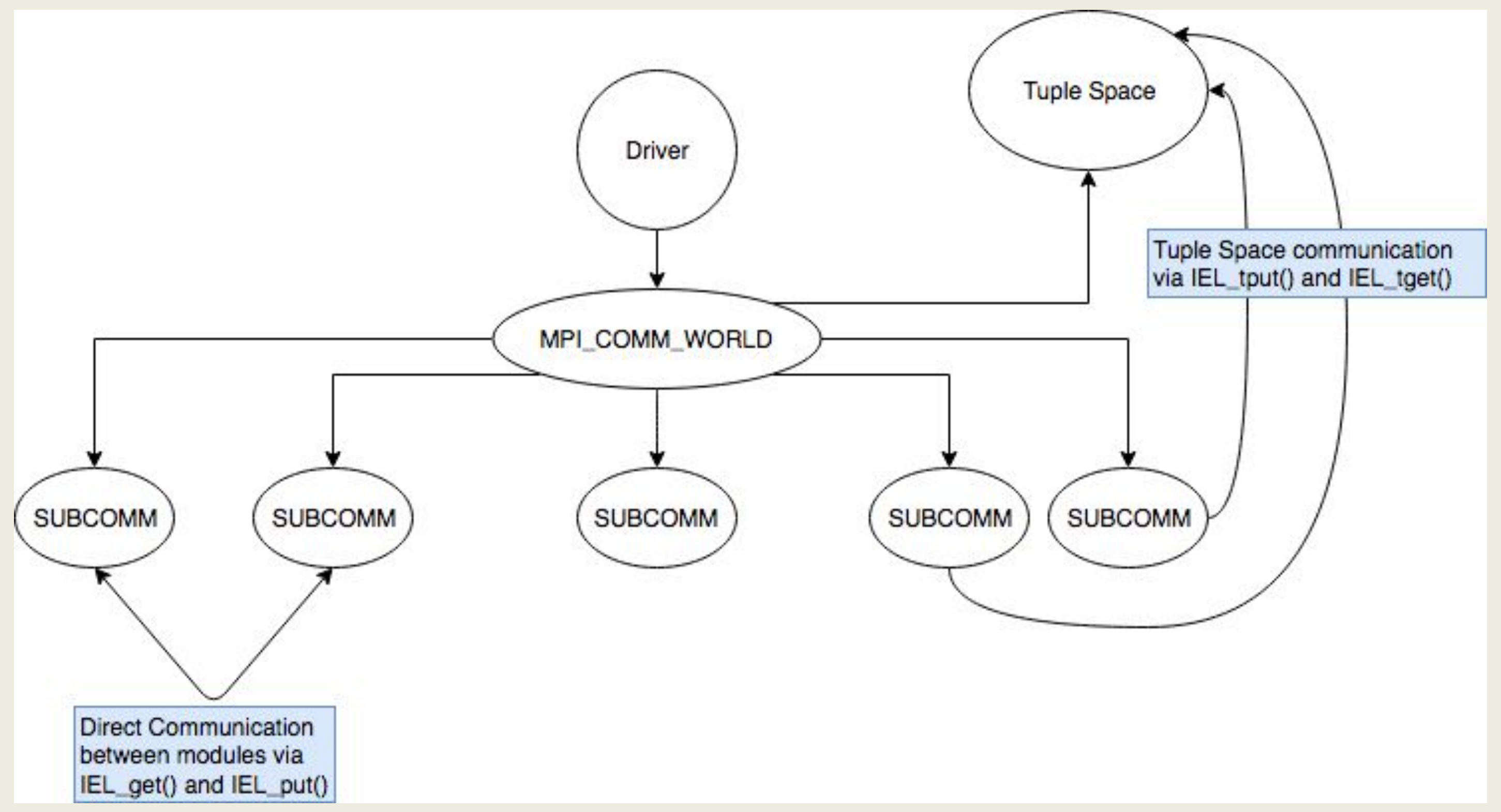
openDIEL: A Parallel Open Source Workflow Engine

Students: Efosa Asemota (Morehouse College), Frank Betancourt (UTK), Quindell Marshall (Morehouse College), Mentor: Dr. Kwai Wong (UTK)

Introduction

Open Distributive Interoperable Executive Library (openDIEL) is workflow engine intended to launch batch jobs on HPC. openDIEL consists of a set of C code and MPI functions to unify many different modules of computation under a single driver executable.

The basic idea is to create a **configuration file** that with a section that specifies modules you want to run, and another section to specify the **workflow** (i.e. the order in which you want modules to run). Modules can initiate synchronous **direct communication** between modules, or asynchronous **tuple space communication**.



Tuple Space
Driver
Tuple Space communication via IEL_tput() and IEL_tget()
MPI_COMM_WORLD
SUBCOMM
SUBCOMM
SUBCOMM
SUBCOMM
SUBCOMM
Direct Communication between modules via IEL_get() and IEL_put()

Figure 1: Intermodule communication

Modules that run serial code can simply be run with fork() and exec(); these are referred to as **automatic modules**. Modules that run parallel code must be called as a **function** by the driver executable; these are referred to as **managed modules**.

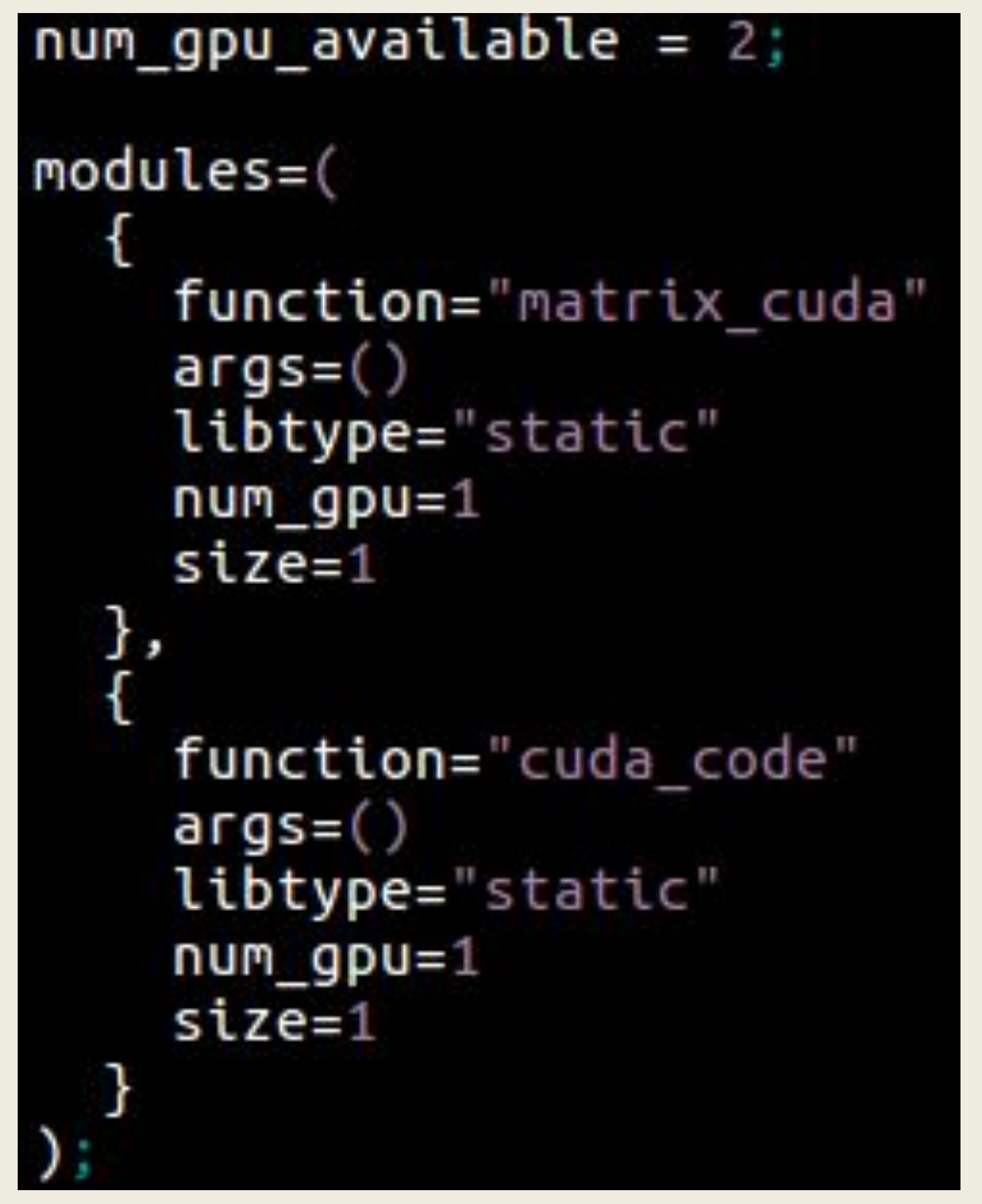
GPU Support

A growing area in supercomputing is the adoption of GPUs as accelerators. In managed modules, openDIEL supports the ability to allocate specific numbers of GPUs on a per-module basis.

Currently, you can specify the number of GPUs you have available on each compute node of the system, and specify how many GPUs will be used by each module.

In the Figure 2, the configuration file specifies two modules, allocating 1 GPU to each.

openDIEL also supports creating multiple different modules that use a combination of CPU, GPU, and multithreading.



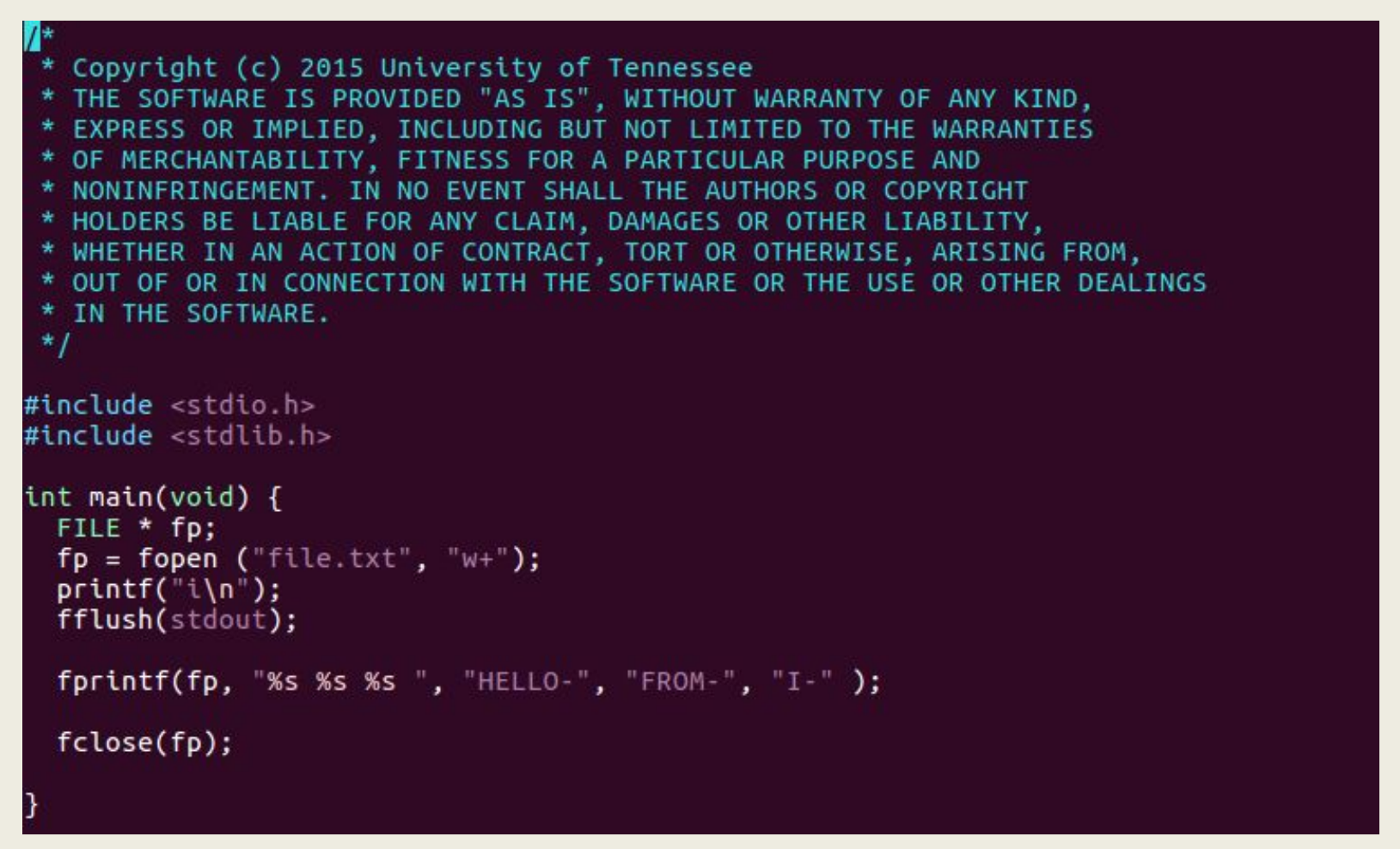
```
num_gpu_available = 2;

modules=(
  {
    function="matrix_cuda"
    args=()
    libtype="static"
    num_gpu=1
    size=1
  },
  {
    function="cuda_code"
    args=()
    libtype="static"
    num_gpu=1
    size=1
  }
);
```

Figure 2: GPU Module Specification in a configuration file

Necessity for a Graphical User Interface

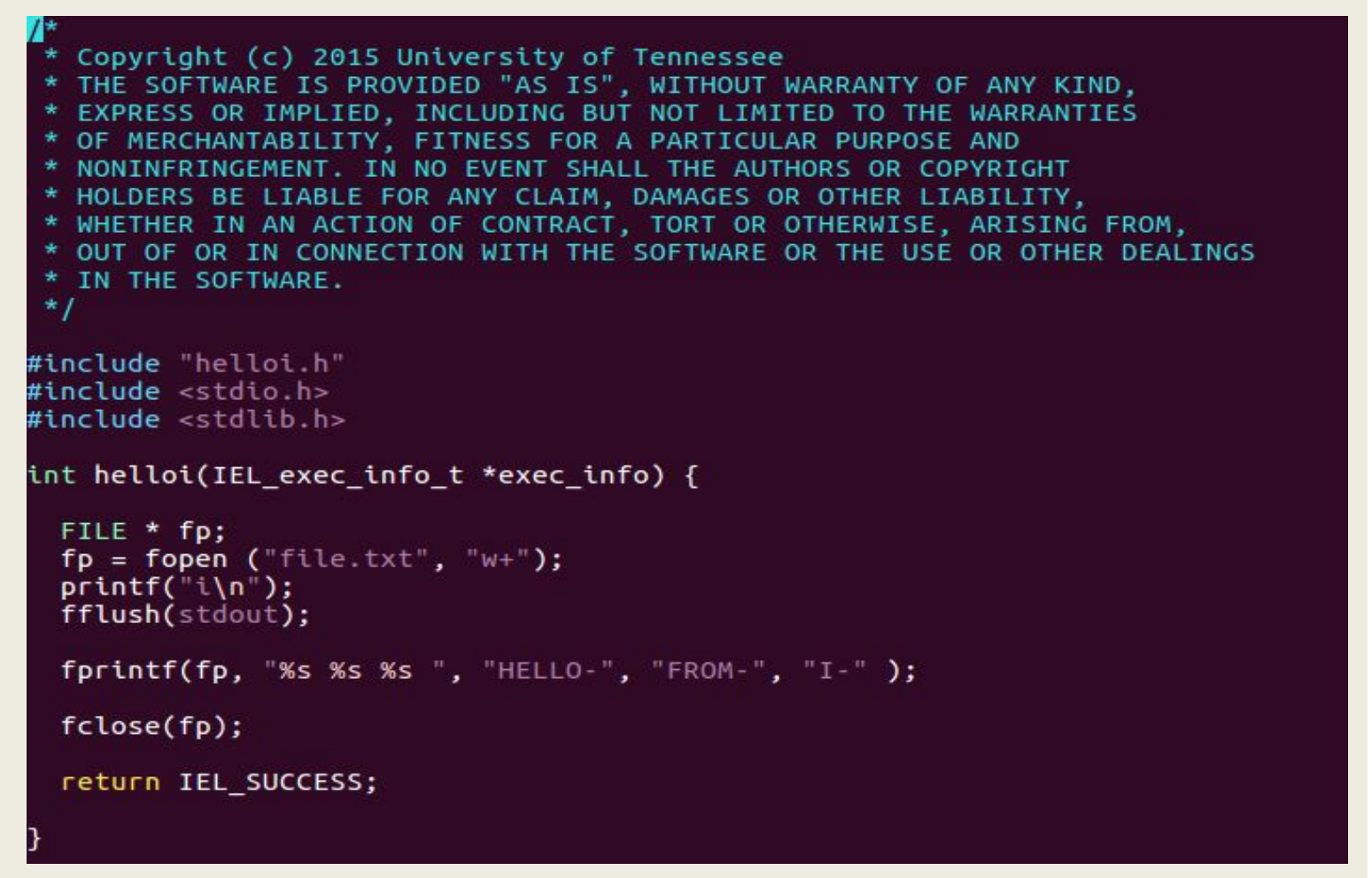
The purpose of the GUI is to provide a much more user-friendly way of utilizing openDIEL. The current process of running modules using openDIEL in managed mode has proven to be very tedious, and the GUI's goal is to handle most of the responsibility for the user.

- One of the first steps that the user would have to do in order use openDIEL is to first convert their code(s) into a module/function using ModMaker.py. Once they've converted their code(s), they would then have to create a header file for their newly formatted module(s).
- After this step is complete, they would then have to create a workflow configuration file, and the purpose of this file is to basically outline and define each module and to then outline how each module will run using openDIEL.
- Next, the user would have edit the Driver.c code, and include each headerfile that corresponds to their modules, and they would also have to go down in the code to the IELAddModule function call and pass as arguments a function pointer to their modules and also the name of that module as a string argument.
- Once all of these steps are complete, they would then have to compile each module as library, and they would also link the libraries to the driver and compile the driver. Lastly, they would need run the driver executable with the workflow configuration file.



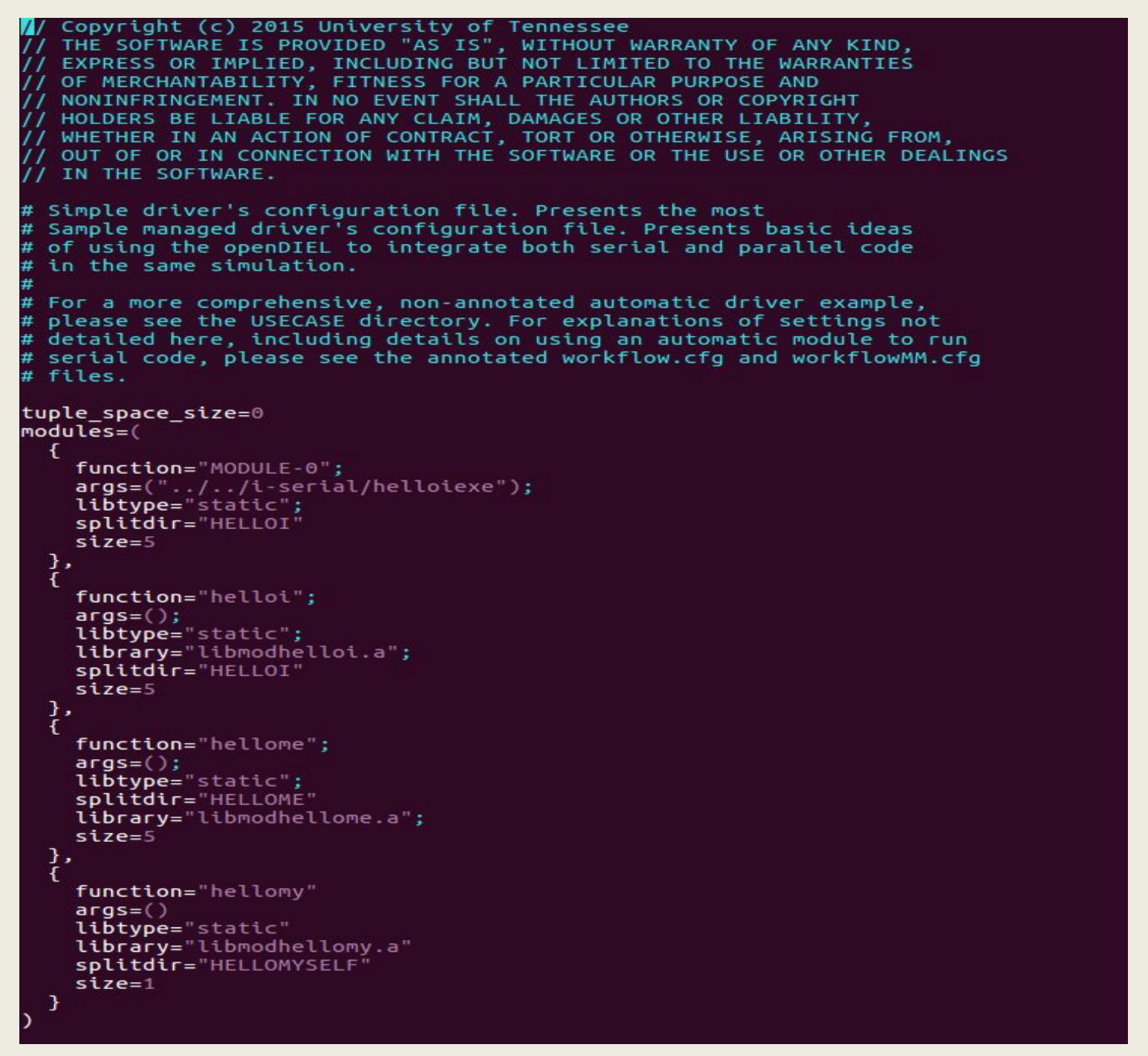
```
/*
 * Copyright (c) 2015 University of Tennessee
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * 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.
 */

#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 3: Original code



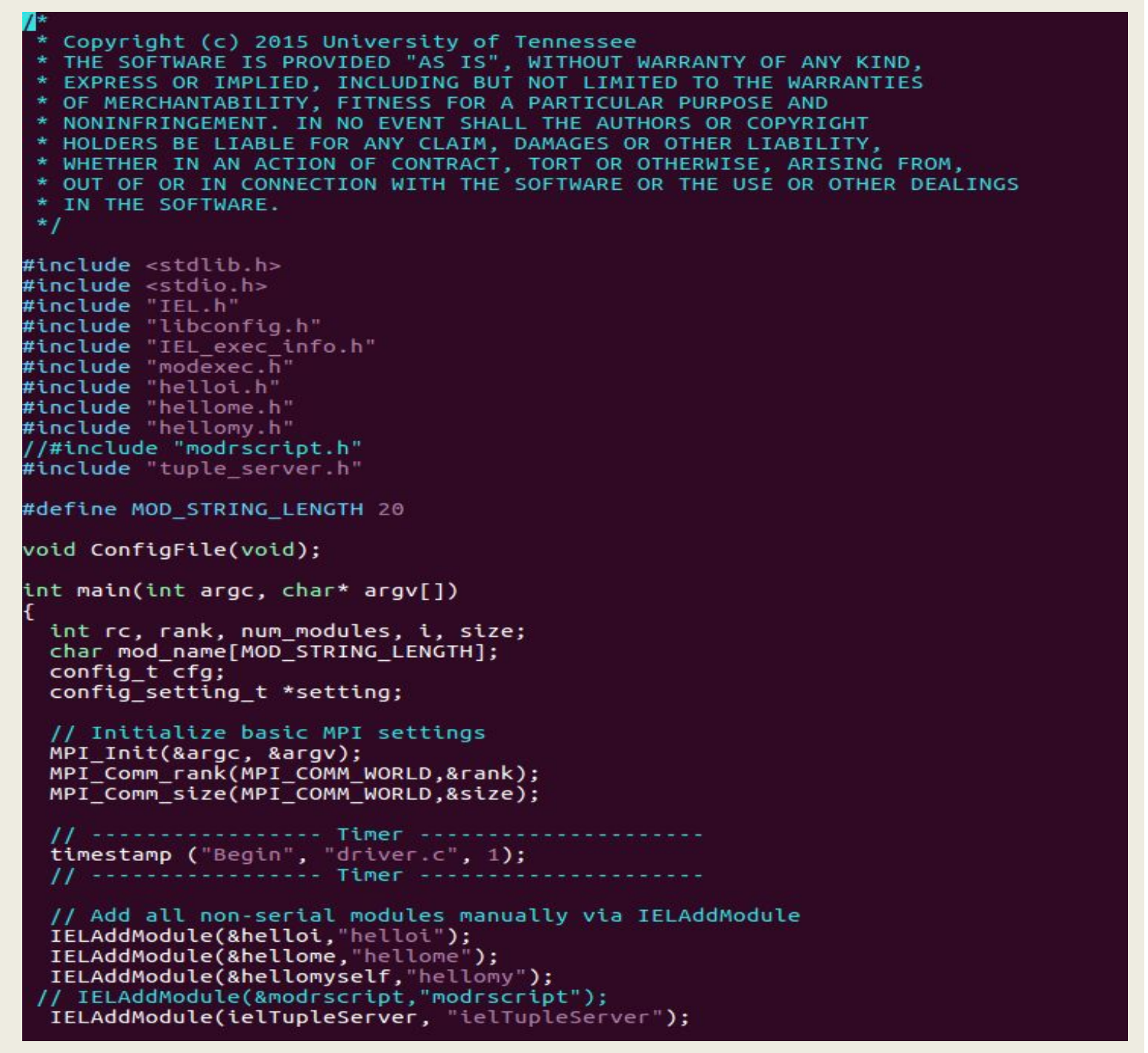
```
/*
 * Copyright (c) 2015 University of Tennessee
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * 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 "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 4: 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.

# 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 5: 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.
 */

#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"

#define MOD_STRING_LENGTH 20

void ConfigFile(void);

int main(int argc, char* argv[])
{
  int rc, rank, num_modules, i, size;
  char mod_name[MOD_STRING_LENGTH];
  config_t cfg;
  config_setting_t *setting;

  // Initialize basic MPI settings
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);

  // ---------------- Timer --------------------
  timestamp ("Begin", "driver.c", 1);
  // ---------------- Timer --------------------

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

Figure 6: Driver.c code

How GUI Solves Issues

The GUI has specific tabs that allow for complete customization:

- The Functions and Attributes tabs are for taking input from the user and storing it as either a function to go into a module, or as a module for the workflow.
- The Workflow and Driver tabs are for planning out the "schedule" of the processes to be done, and for creating the driver to run the code for said processes in module form.

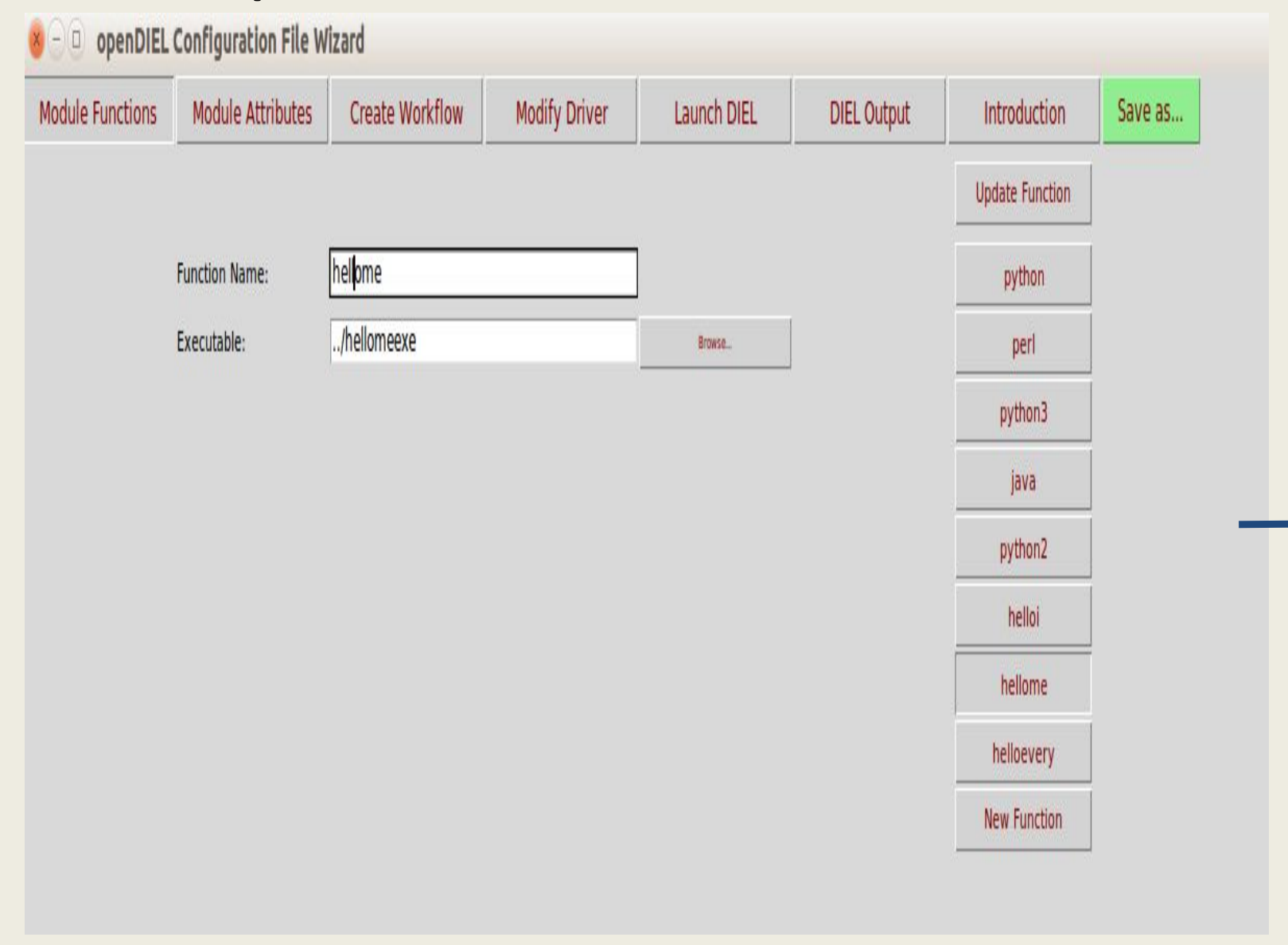


Figure 7: Functions - Storing Functions

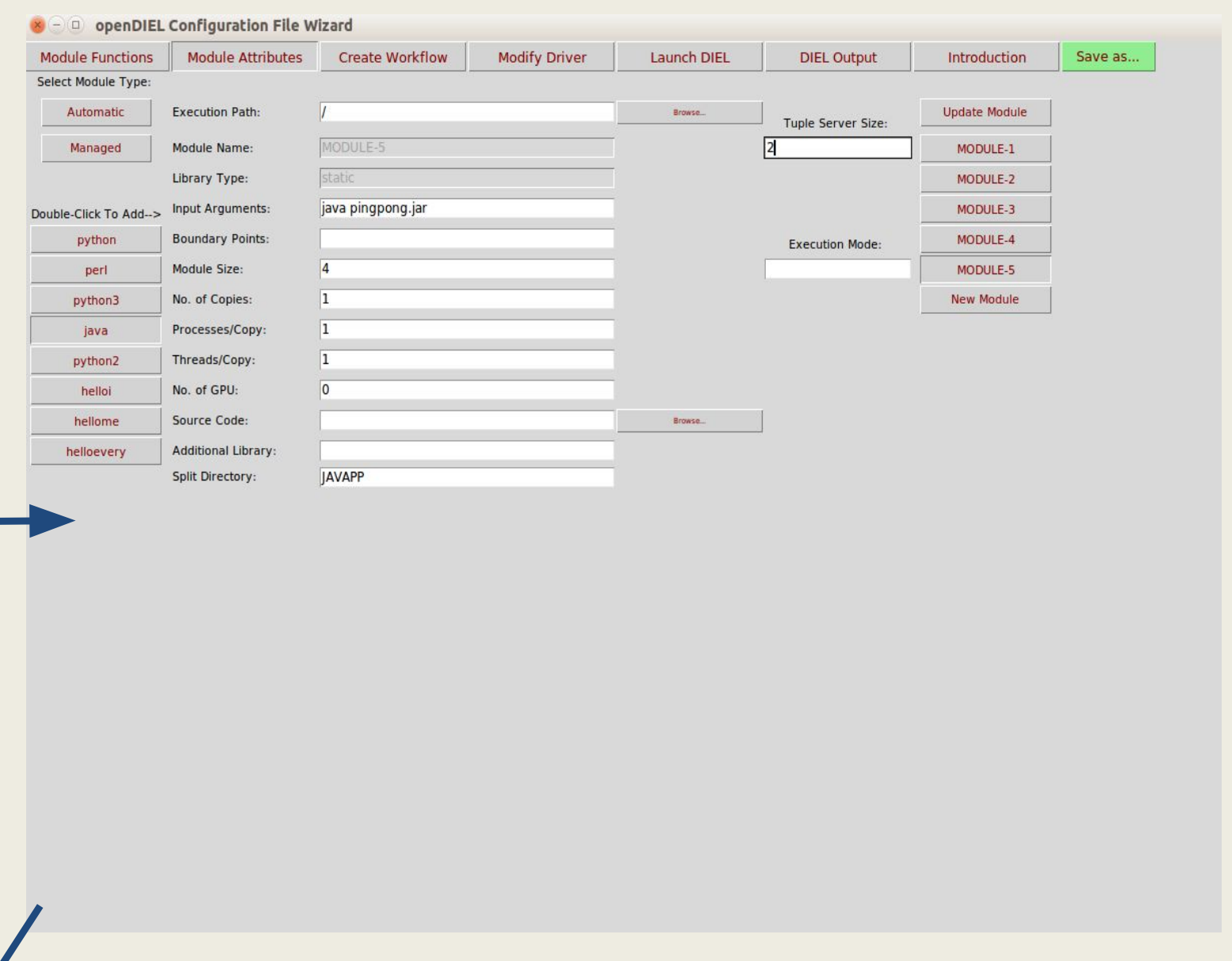


Figure 8: Modules - Module Creation

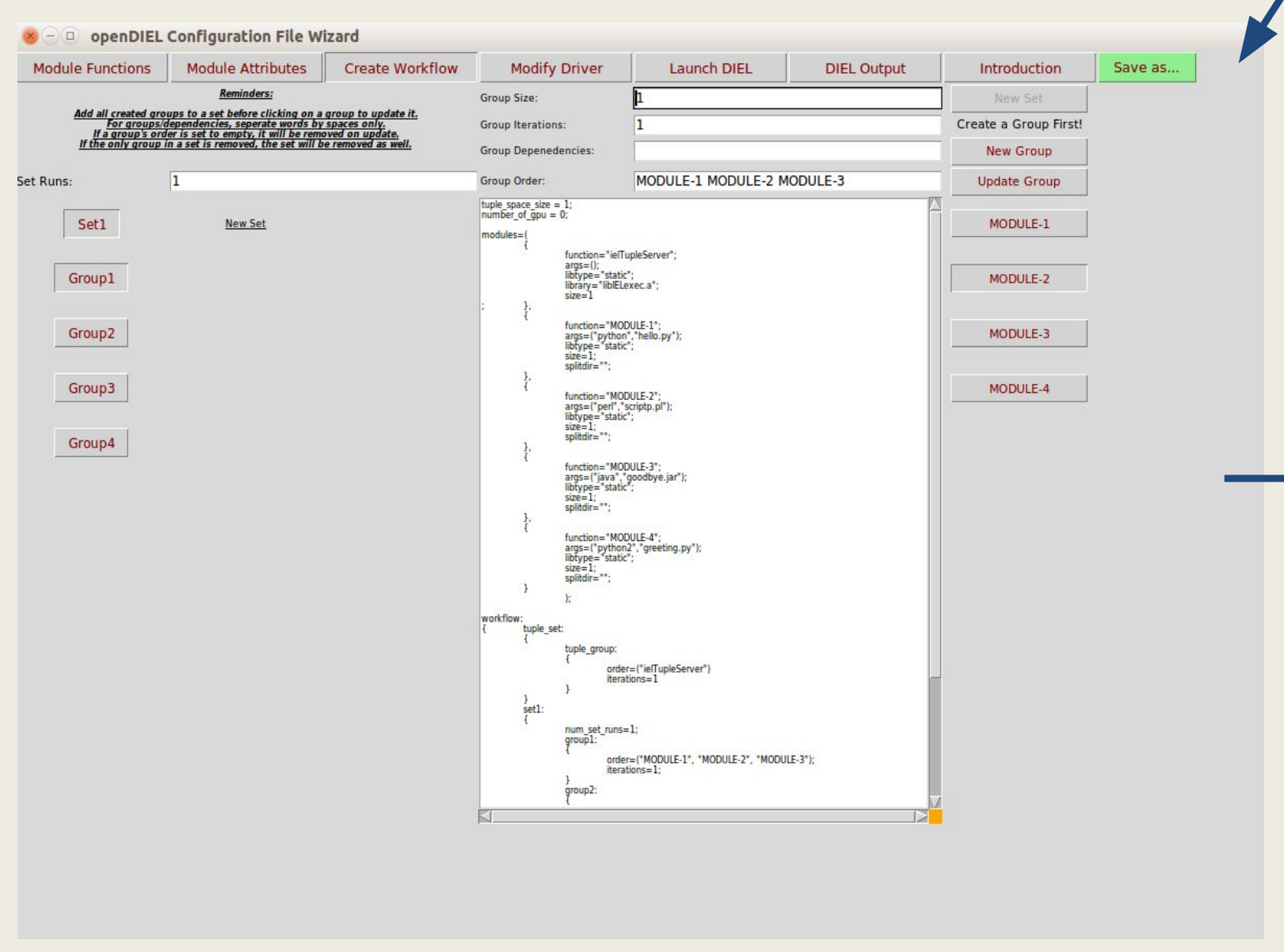


Figure 9: Workflow - Configuration File

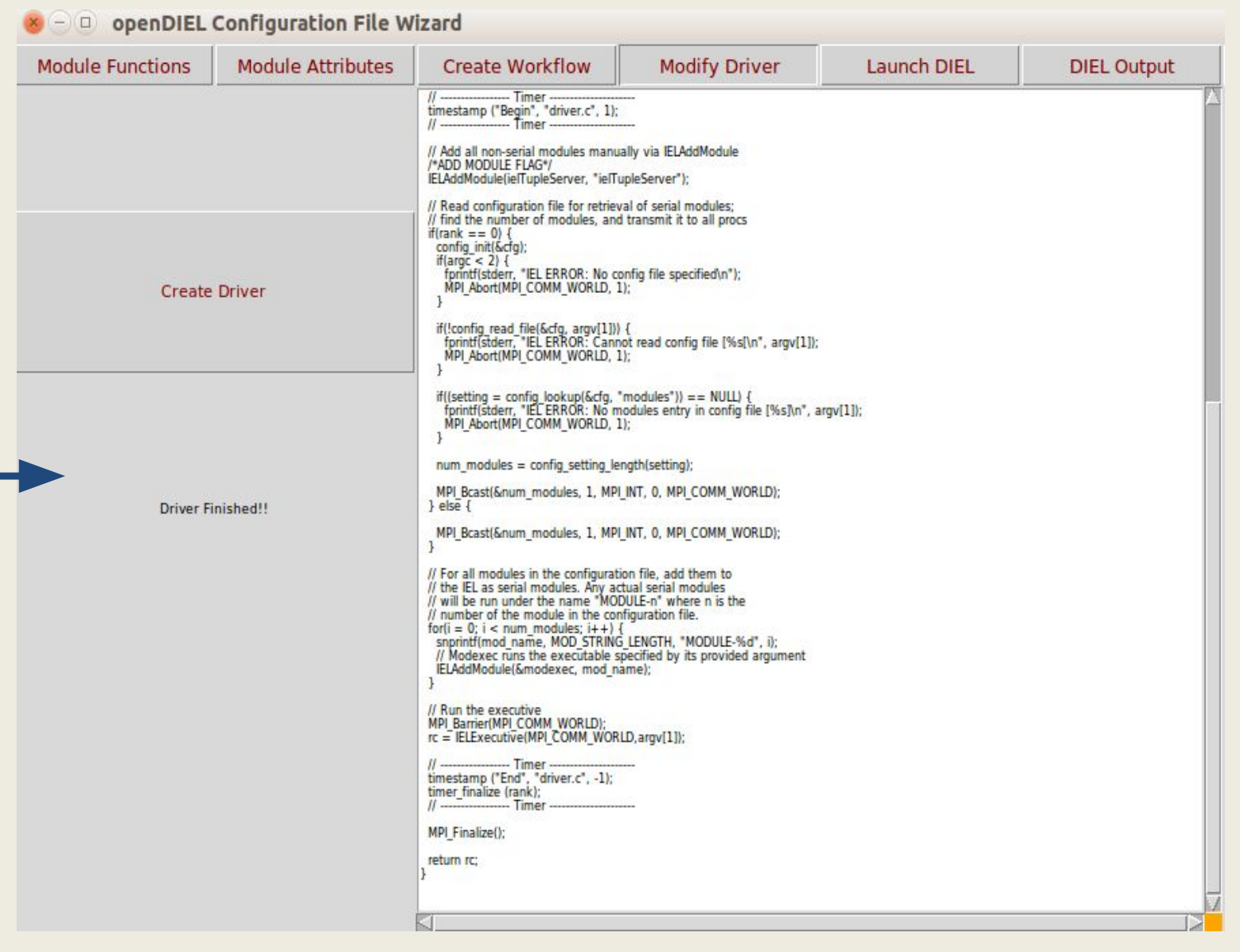


Figure 10: Driver - New Driver Code

Future Work

- Making the GUI usable on supercomputers, by adding the ability to automatically generate SLURM batch scripts for running jobs on XSEDE supercomputers Comet and Bridges.
- Add the ability to automatically build managed modules for molecular dynamics software LAMMPS, with support for compiling the GPU and CPU versions
- Additional testing, validation, and documentation of previously implemented features

Acknowledgements

This project was sponsored by the National Science Foundation through Research Experience for Undergraduates (REU) award, with additional support from the Joint Institute of Computational Sciences at University of Tennessee Knoxville. This project used allocations from the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by the National Science Foundation. In addition, the computing work was also performed on technical workstations donated by the BP High Performance Computing Team.

[1] K. Wong, L. Brown, J. Coan, D. White, *Distributive Interoperable Executive Library (DIEL) for Systems of Multiphysics Simulation.* (2015)