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

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

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

What?

- openDIEL is a wrapper to schedule work on a set of resources.

- Primarily intended as a workflow engine to launch batch jobs on HPC

- Consists of a set of C code, and MPI functions to make it work

- The idea is a create a single MPI executable, and specify how processes will run in a configuration file

- Communication takes place between modules, which are split into separate sub-communicators under the MPI_COMM_WORLD created by the main driver

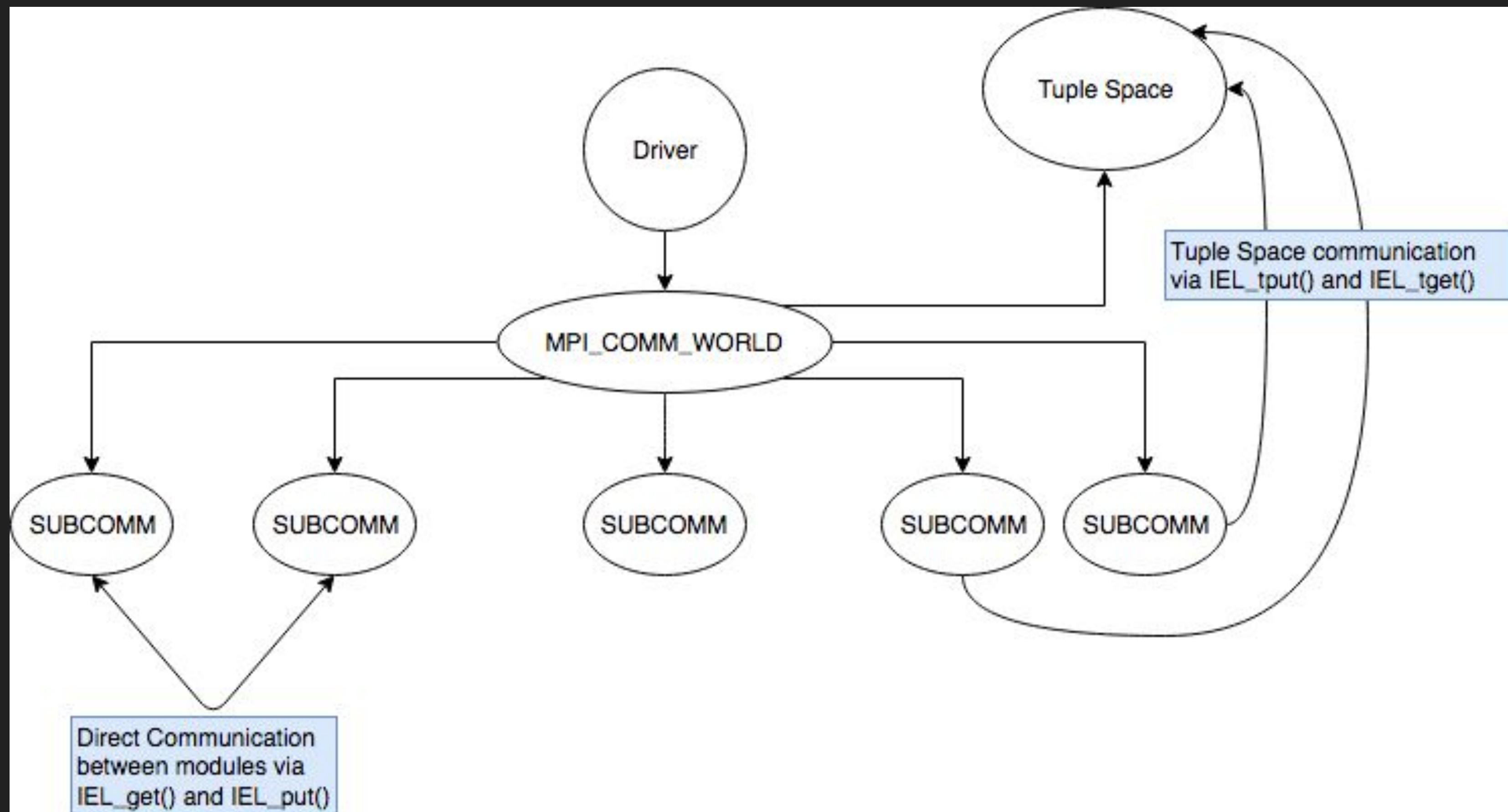

Figure 1: Graph of module communication

Configuration Files

```
modules=(                                # <- This section give information about modules
  {
    function="MODULE-0";                 # <- Serial code, can be specified
    args=("../i-serial/helloiexe");      #    as an automatic module, and
    libtype="static";                    #    run by modexec with fork()
    splitdir="HELLOI";                   #    and execvpe()
    size=5
  },
  {
    function="alone";                    # <- Parallel code, it is specified as
    args=();                             #    a managed module, meaning it must
    libtype="static";                    #    be compiled and linked as an external
    exec_mode="parallel";                #    library to the driver, and then called
    copies=2;                            #    as a function
    processes_per_copy=3
    size=6
    threads_per_process=1
    splitdir="ALONE-MOD"
  }
```

Figure 2: Example Configuration File

```
workflow:                        # <- The workflow section contains info on
{                                #    on how the previously specified module
  set1:                          #    are to be run
  {
    group1:
    {
      order=("alone")
      iterations=1
    }
  }
  set2:
  {
    group1:
    {
      order=("MODULE-0")
      iterations=1
    }
  }
}
```

Addition of GPU Keywords

No way to specify options for jobs that use GPU

- Create keywords that allow the usage of GPU
- Be able to prescribe how modules will utilize GPUs

```
modules=(
  {
    function="gpu-code";
    args=();
    libtype="static";
    exec_mode="parallel";
    num_gpu=4
    size=2
    gpu_per_process=2
  },
  {
    function="more-gpu-code"
    args=()
    libtype="static"
    exec_mode="parallel"
    num_gpu=2
    gpu_per_process=2
    size=1
)
```

Figure 3: How GPU keywords might look

Why do we need a Graphical User Interface (GUI)?

- The purpose of the GUI is to provide a more user-friendly way of using OpenDIEL.
- In order to currently run programs using OpenDIEL, the user would have to go through a lot of steps.
- One of the first steps that they would have to do is to convert their programs into functions using ModMaker.py
- The next step would be for them to create a header file using their newly formatted code and to also create and compile it as a library.
- The next step would be for them to go into workflow configuration file and make and add all of the necessary changes that they would need in order run the code.

Why do we need a Graphical User Interface (GUI)? (contd.)

- The next step would be for them to go into the DriverMM.c code, and include the all of their codes as header files.
- They would then also go down to the IELADDModule function call, and pass as arguments (&nameOfCode, “name of code”)
- They would also need to go through several Makefiles and make necessary corrections in order to run the code.
- With the GUI, we are hoping to negate the need of the user to go through all of these tasks, and to instead have the GUI handle all of the responsibility for them.

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 4: This is a simple example of a Hello World program before it has been converted by ModMaker.py

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

Header 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 "IEL_exec_info.h"

#ifndef _MAINMOD_helloi_H
#define _MAINMOD_helloi_H

int helloi(IEL_exec_info_t *exec_info);

#endif
~
~
```

Figure 6: Example of a Header 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 7: Configuration File

```
workflow:
{
  groups:
  {
    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 8: Another part of the Configuration File

```c
/*
 * 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 9: Driver Code

```
#######################################################
#
#    Makefile for HELLOWORLD
#
#####################################################

all:
	cd i-serial;make ; cp *exe ../DRIVER
	cd i; make; cp *.a ../MODULE-FILE
	cd me; make; cp *.a ../MODULE-FILE
	cd myself; make; cp *.a ../MODULE-FILE
	cd MODULE-FILE; make
	cd DRIVER; make
	cd USECASE/WORKFLOW-AM; cp ../../DRIVER/*exe .
	cd USECASE/WORKFLOW-AM; cp ../../DRIVER/driverAM .
	cd USECASE/WORKFLOW-MM; cp ../../DRIVER/*exe .
	cd USECASE/WORKFLOW-MM; cp ../../DRIVER/driverMM .

cleanall:
	cd i-serial;  make clean
	cd i; make clean
	cd me; make clean
	cd myself; make clean
	cd MODULE-FILE; make clean
	cd DRIVER; make clean
~
~
```

Figure 10: Makefile

```
IEL_HOME=../../..
include $(IEL_HOME)/Makefile.inc

CFLAGS    = -Wall

CLIBS     = -lm  -lz -ldl

INCLUDES=$(MACH_INC)

static: libmodhelloi.a

libmodhelloi.a: helloi.c
	$(MPICC) -c $(INCLUDES) helloi.c
	ar -rcs libmodhelloi.a helloi.o

clean:
	rm -f *.o *.a
~
```

Figure 11: Makefile

```
#Path to top level of openDIEL
IEL_HOME=../../..
#Include the global Makefile.inc
include $(IEL_HOME)/Makefile.inc

CFLAGS   = -Wall

OMP      = -fopenmp

CLIBS-SM = -L../MODULE-FILE -lmodhelloi -lmodhellome -lmodhellomy
CLIBS-MM = -L../MODULE-FILE -lmodhelloi -lmodhellome -lmodhellomy -lmodexec
CLIBS-AM = -L../MODULE-FILE -lmodexec
##CLIBS-R = ../MODULE-FILE/libmodrscript.a -dynamic $(R_LIB)
CLIBS-R =

CLIBS    = -lm  -lz -ldl

INCLUDES=$(MACH_INC) -I../MODULE-FILE
INCLUDE-R=$(R_INC)

LDFLAGS=$(MACH_LIB) $(LIBCONFIG_LIB) $(CLIBS)

all:    driverAM driverSM driverMM

driverSM:
        $(MPICC) $(CFLAGS) $(INCLUDES)  -c driver.c
        $(MPICC) $(OMP) -o driverSM driver.o  $(LDFLAGS) -fopenmp $(CLIBS-SM)

driverMM:
        $(MPICC) $(CFLAGS) $(INCLUDES)  $(INCLUDE-R) -c driverMM.c
        $(MPICC) $(OMP) -o driverMM driverMM.o  $(LDFLAGS) $(CLIBS-MM) $(CLIBS-R)

driverAM:
        $(MPICC) $(CFLAGS) $(INCLUDES)  -c driverAM.c
        $(MPICC) $(OMP) -o driverAM driverAM.o  $(LDFLAGS) $(CLIBS-AM)
clean:
        rm -rf *.o driverSM driverAM driverMM *exe Timers HELLO*
~
~
```

Figure 12: Driver Makefile

The Graphical User Interface

- Major Areas of GUI:
  - Module Functions and Attributes
  - Workflow and Drivers
  - Launch & Output

Module Functions and Attributes

Module Functions: Serial Code Storage

- Holds name and pathway of function

Module Attributes: Module description

- General Information for Module, including current working directory
- Allows user to search for programs, function libraries, and other files

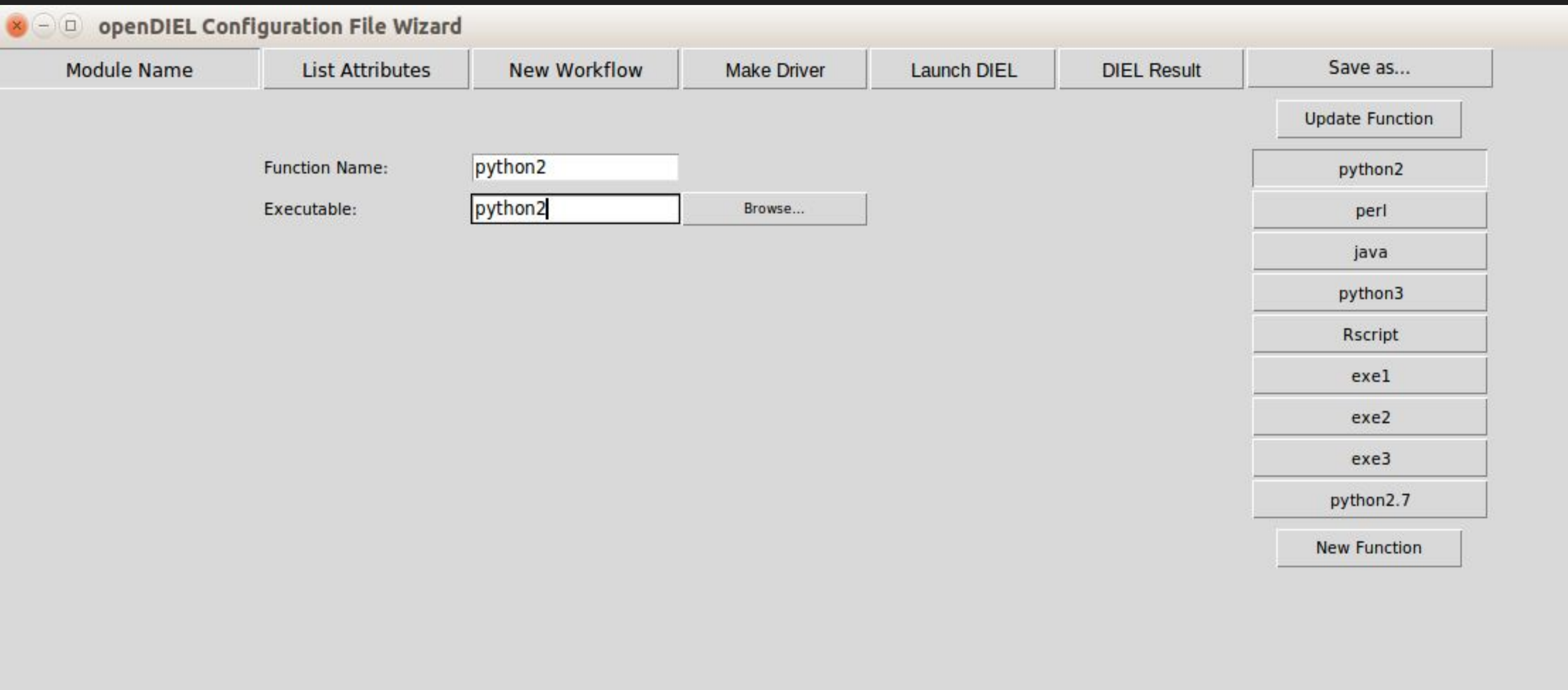

Figure 13: “Module Name” Tab

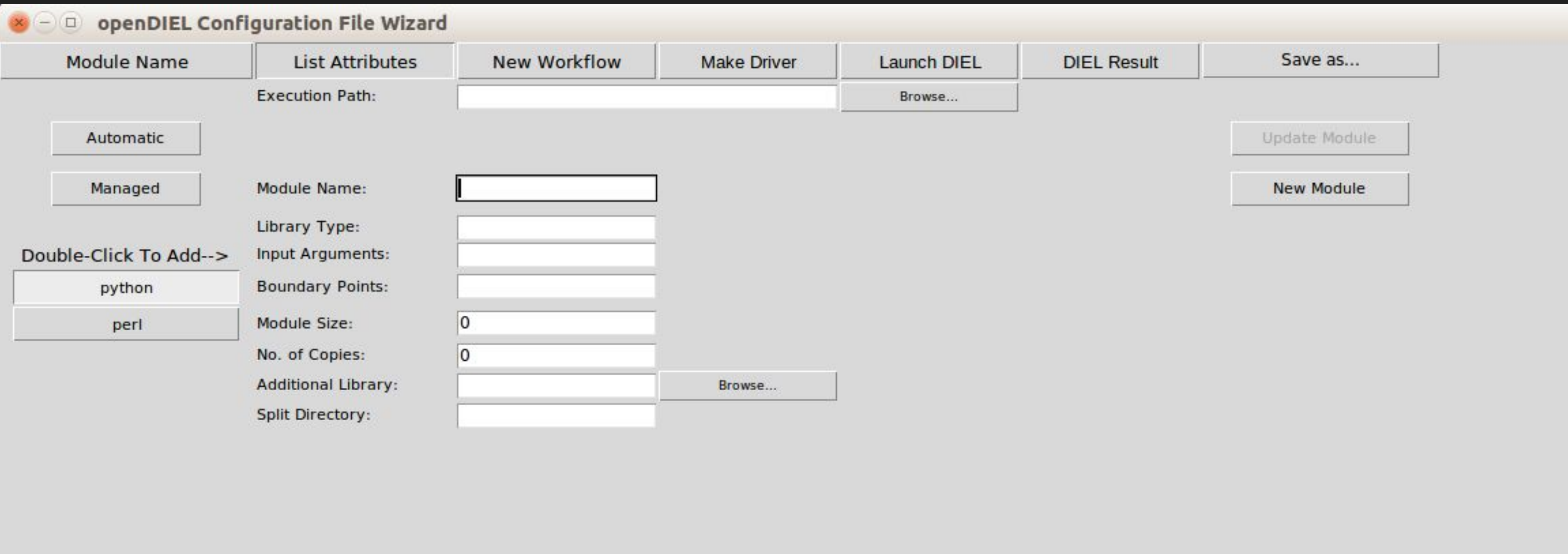

Figure 14: "List Attributes" Tab

Workflow and Driver

Workflow: Layout of modules, ordering, processor requirement, etc.

- Creates arrangement of groups for running, including number of copies to be run, as well as number of iterations for each ordering.

Driver: Automatic and Manual Drivers

- Automatic: Used when all files are *source code,* code is run in specified order without changes with locked module names
- Manual: Used to handle *parallel code* files.

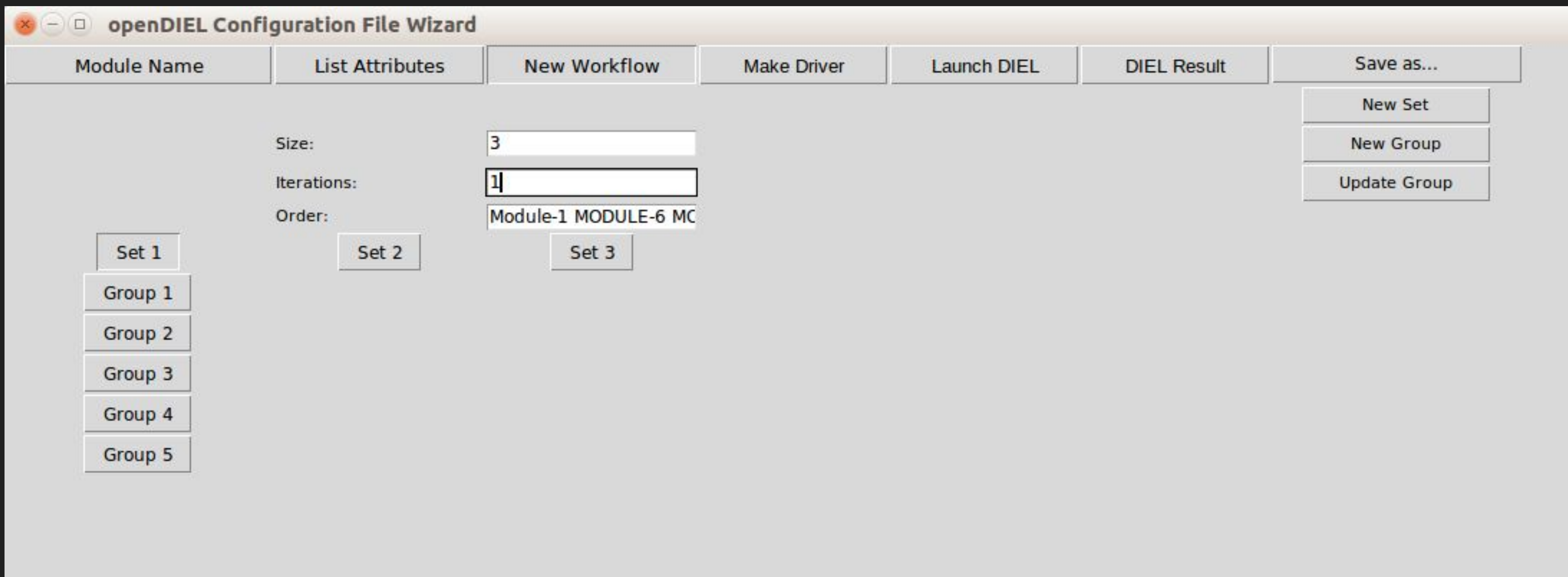

Figure 15: "New Workflow" Tab

Launch And Output

Launch(Future): Launches the workflow engine with the selected options and given data.

- User can decide number of processors, method, etc.

Output(Future): Presents the result of the workflow engine in its own tab.

Well, that’s it!

Q&A