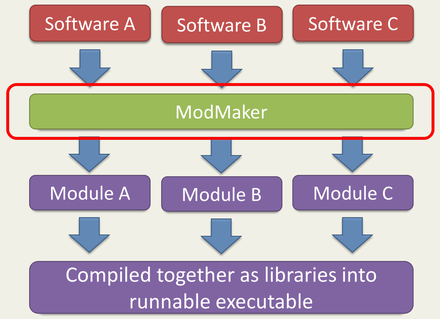
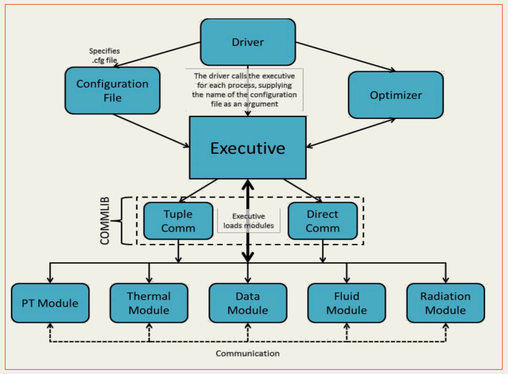
User Interfaces Development in openDIEL

Argens Ng (HKU), Tanner Curren (Maryville College)
Mentor: Kwai Wong (UTK)

What is openDIEL?

OpenDIEL stands for **open Distributive Interoperable Executable Library**. It aims at facilitating cooperation between loosely coupled modules under the **MPI** (Message Parsing Interface) system. With user-defined configuration file and driver, openDIEL can output a **single executable** doing the work of multiple modules, which can be a crucial function in scenarios where users have to conduct microsimulations in very small time intervals side by side.





What is the problem?

Notice that openDIEL takes in **modules** for its operation. These have to follow the format of openDIEL and users had to conduct changes in their source code accordingly. In terms of modularity and encapsulation, this practice is highly problematic and hence the need for enhancements becomes apparent..

What enhancements are made?

There are four main enhancements. The first one is a module maker, or **modMaker**. It scan the user's code and suggest transformation under the formatting rules of openDIEL. The second one is the addition of **timer**, which we hope can be both used for early-stage as well as late-stage development. The third one is **I/O improvement** and last **Fortran** support, as we believe that many antique codes are written in Fortran and they are still used in many simulation models.

ModMaker – Workflow

To conduct module transformation, we look into the syntax of C. Basically every statement, declaration, variable has its **identifying feature**. For example "int" followed by spaces, tabs then "main" represents the main program. This pattern can only exist once per program (or twice if we separate declaration and implementation).

However, while we can identify target strings with identifying feature, we almost always need to change more than just it. Difference between identifying feature and **replacement candidate** would be explained in **Syntactic Freedom**.

We then prompts the user confirmation for any change, because we believe the user knows the most about their own code.

ModMaker – Pattern Matching

ModMaker works off the principle of **pattern matching** of strings. It basically treats a C program as a document line by line, then attempts to locate "identifying features", or target pattern. Earlier attempts tried to conduct multi-degree string search in one scan, ie. For each character, attempt to:

(1) see if it is the start of any target string; and
(2) see if it is the continuation of candidate target string.

However we soon realize that reduction of time complexity is impossible without human optimization. This then gives the same complexity as string wise search, with ***n** = total character of string list* & ***s** = total character of document*:

$$O(n, s) = ns$$

This is because information is ultimately proportional to number of comparisons. Looking for individual target string in a document one-by-one is no different from matching each character for all strings.

Text	/	*	T	h	i	s		i	s
Pattern 1	i	n	t		m	a	i	n	
Pattern 2	M	P	I	_	I	n	i	t	
No match x 2	/	*	T	h	i	s		i	
No match x 2		*	T	h	i	s		i	s

Fig 1. Attempting to do all comparisons in one iteration of document

Text	/	*	T	h	i	s		i	s
Pattern 1	i	n	t		m	a	i	n	
No match	/	*	T	h	i	s		i	
No match		*	T	h	i	s		i	s
Pattern 2	M	P	I	_	I	n	i	t	
No match	/	*	T	h	i	s		i	
No match		*	T	h	i	s		i	s

We cannot get more information from same number of comparison. We can only waste less comparison, which is impossible to automate in this case.

Fig 2. Focusing on each target pattern one at a time

ModMaker – Syntactic Freedom

When transforming modules, syntactic freedom was both a nuisance and convenience. It is a challenge for both identifying and modifying.

```
1) int main(int argc,char**argv){\n
2) MPI_Init(&argc,_&argv);
3) rc=MPI_Init(&argc,&argv);
4) int rc=MPI_Init(&argc,&argv);
5) int main(intargc,char**argv) {\n
```

Locating replacement candidate
In (1), we can clearly see that while "int main" is the identifying feature, the declaration of argument also needs to be replaced. The light-underlined segment is the "replacement candidate".

Spaces
Highlighted in (2) and (5) are the importance of spaces. While humans can clearly the difference, interpreters require more rules to determine if a space is crucial or not. Also we need to deal with indefinite spaces and tabs between tokens.

Variable Declaration
As we can see in (2), (3) and (4), function returns can be either abandoned, used, or used to declare. This translates into 3 forms of handling, which ultimately results in a more complex system.

SOLUTION – modularization
To combat all these issues, ModMaker utiilzes a very modular approach to tackle the problem. Three basic categories (Function Title, Statement, Variable) were set to allow different settings, or different expectation of syntactic freedom when dealing with replacement candidates. New exceptions which need to be handled can then be quickly implemented.

Future Work

Timer A timer would be implemented to facilitate the use of both testing and optimization. We plan to provide the time, efficiency as well as improvements as a result of utilizing openDIEL, which ultimately should lead to better optimization in future development as well as for future users.

Multiple I/O This would be more of an actualization of visions, as we enable multiple copies of parallel modules to input and output into separate folders. Unlocking full potential of openDIEL and greatly enhancing user experience by fulfilling their expectation.

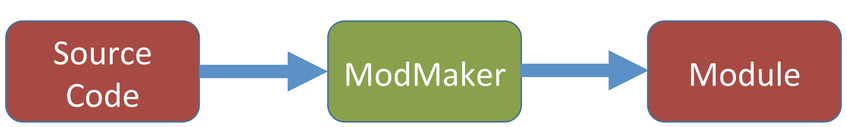
ModMaker – Fortran support Fortran has a much stricter formatting rules than C which is very different form the origin. However, as openDIEL gains better capability at handling Fortran code, ModMaker has to keep up to provide better and all-rounded user experience.

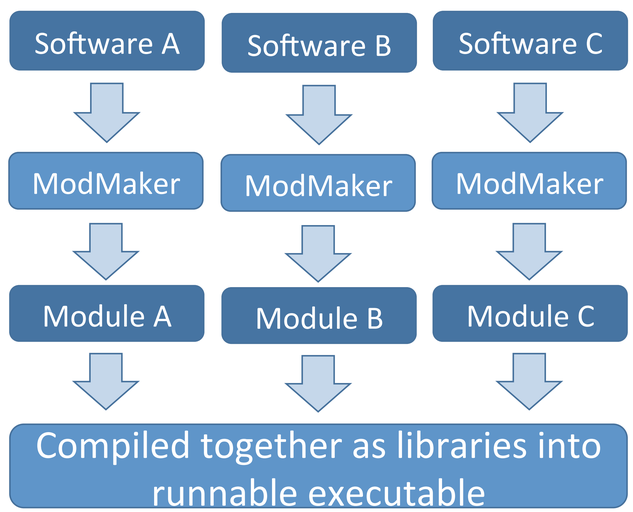
Acknowledgements

This project is made possible only with the support by our menor Dr. Wong, the NSF, the University of Tennessee and Oak Ridge National Laboratory.

Text	/	*	T	h	i	s		i	s
Pattern 1	i	n	t		m	a	i	n	
Pattern 2	M	P	I	_	I	n	i	t	
No match x 2	/	*	T	h	i	s		i	
No match x 2		*	T	h	i	s		i	s

Text	/	*	T	h	i	s		i	s
Pattern 1	i	n	t		m	a	i	n	
No match	/	*	T	h	i	s		i	
No match		*	T	h	i	s		i	s
Pattern 2	M	P	I	_	I	n	i	t	
No match	/	*	T	h	i	s		i	
No match		*	T	h	i	s		i	s





Text	/	*	T	h	i	s		i	s		a		C		p	r	o	g	r	a	m	*	/	i	n	t		m	a	i	n
Pattern 1	i	n	t			m	a	i	n																						
Pattern 2	M	P	I	_	I	n	i	t																							
No match x 2	/	*	T	h	i	s		i																							
No match x 2		*	T	h	i	s		i	s																						
No match x 2			T	h	i	s		i	s																						

Text	/	*	T	h	i	s		i	s		a		C		p	r	o	g	r	a	m	*	/	i	n	t		m	a	i	n
Pattern 1	i	n	t			m	a	i	n																						
No match	/	*	T	h	i	s		i																							
No match		*	T	h	i	s		i	s																						
No match			T	h	i	s		i	s																						
Pattern 2	M	P	I	_	I	n	i	t																							
No match	/	*	T	h	i	s		i																							
No match		*	T	h	i	s		i	s																						
No match			T	h	i	s		i	s