OpenDIEL

Supported by The National Science Foundation

Tristin Baker, Jordan Scott, and Zachary Trzil
Mentor: Dr. Kwai Wong

What is the OpenDIEL?

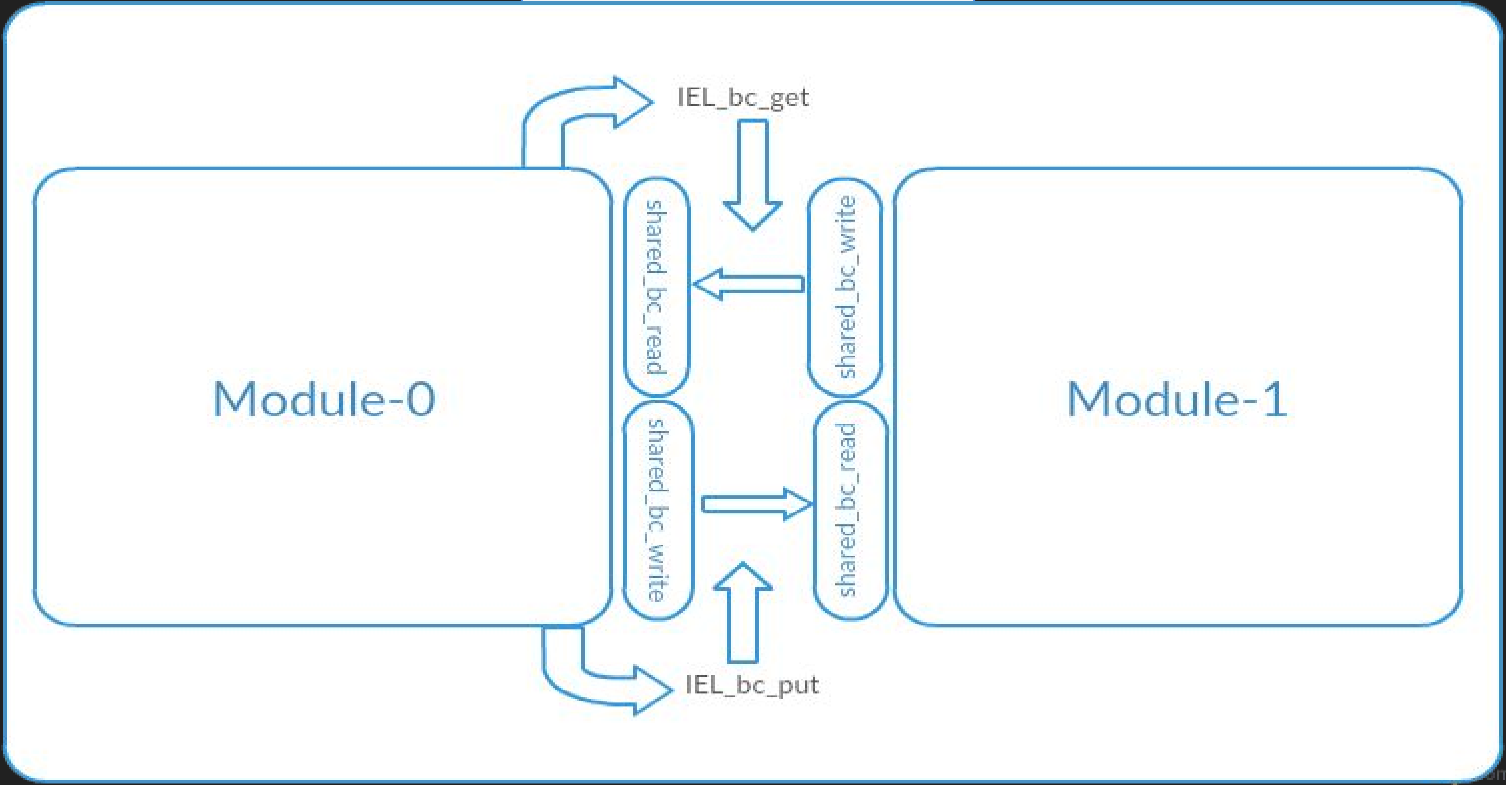
- Framework for running multiple parallel softwares as OpenDIEL Modules
  - Allows for communication and data transfer between the different modules
- Intended for use with high performance computing
  - Use intended for multiple process softwares which require transfer of a large amount of data
- Uses MPI (Message Passing Interface) to facilitate transfer of data and information the different modules need
- Uses a driver to drive the IELExecutive
  - Driver reads a workflow file to identify the different modules/groups/sets the IELExecutive will use in order to run
- Has two forms of communication
  - Direct Communication and Tuple Space Communication.

Direct Communication

- Method for modules to share data directly between one another
- Facilitated by a shared boundary condition
  - Main method of direct communication
  - Found in IEL_exec_info_t data structure as double * shared_bc
    - Each module has shared_bc_write and shared_bc_read
      - Sizes modified in workflow configuration file
  - shared_bc size is set in the workflow configuration file
- IEL_bc_put and IEL_bc_get
  - Method the modules use for their shared_bc_read and shared_bc_write to communicate
  - Wrappers around MPI_Send and MPI_Receive
  - IEL_bc_exchange

Direct Communication Visualized

shared_bc

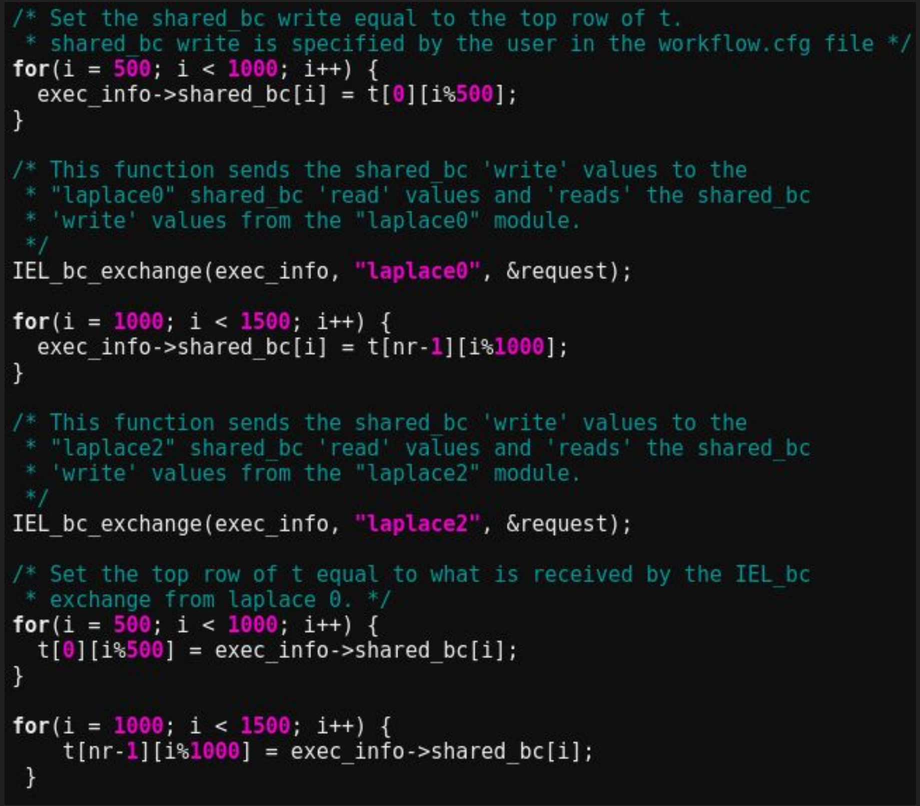


Using Direct Communication For Laplace Example

- OpenDIEL implementation of MPI Laplace example program
- In the original, the 1000x1000 matrix existed in one function.
- With the OpenDIEL implementation, the matrix can be any size and can be split into however many functions the user wants thanks to the shared_bc.
- Within the workflow configuration file, the user sets where the modules can read from and write to.
    - In this example, the user will want the ‘shared_bc_write’ to be equal to its top and bottom rows, and its ‘shared_bc_read’ field to be able to read from the module above and below’s ‘shared_bc_write’ field.

Laplace Continued

- Laplace-1
  - Middle module in this example
- In this screenshot, shared_bc is set to the top row of laplace-1
  - This is sent to laplace-0 via IEL_bc_exchange
    - This function call also receives laplace-0's shared_bc_write field and stores it in laplace-1's shared_bc_read field
  - Once data is received, it is stored in the proper row of t (laplace-1's matrix)
- This process is repeated for laplace-2.



```
/* Set the shared_bc write equal to the top row of t.
 * shared_bc write is specified by the user in the workflow.cfg file */
for(i = 500; i < 1000; i++) {
  exec_info->shared_bc[i] = t[0][i%500];
}

/* This function sends the shared_bc 'write' values to the
 * "laplace0" shared_bc 'read' values and 'reads' the shared_bc
 * 'write' values from the "laplace0" module.
 */
IEL_bc_exchange(exec_info, "laplace0", &request);

for(i = 1000; i < 1500; i++) {
  exec_info->shared_bc[i] = t[nr-1][i%1000];
}

/* This function sends the shared_bc 'write' values to the
 * "laplace2" shared_bc 'read' values and 'reads' the shared_bc
 * 'write' values from the "laplace2" module.
 */
IEL_bc_exchange(exec_info, "laplace2", &request);

/* Set the top row of t equal to what is received by the IEL_bc
 * exchange from laplace 0. */
for(i = 500; i < 1000; i++) {
  t[0][i%500] = exec_info->shared_bc[i];
}

for(i = 1000; i < 1500; i++) {
    t[nr-1][i%1000] = exec_info->shared_bc[i];
 }
```

To Do:

- Develop proper documentation
  - Previously, this project was very poorly documented, so much of the initial 4 weeks of this program were spent trying to decipher what was happening in the code. We hope to sufficiently document the code so that people that work on this in the future will have an easier time developing, and users of the OpenDIEL will be able to see which functions do what.
- Remove Global shared_bc
  - In its current implementation, if direct communication is being used, OpenDIEL designates a shared_bc array of size num_shared_bc to each module. This is a waste of memory for the most part, and is not scalable. If the num_shared_bc is 10000 and a module only accesses 500 elements from that array, there are 9500 unused elements. This problem only expands as you scale up the num_shared_bc size.

Tuple Space Communication

- Associative memory paradigm for distributed/parallel computing
  - Repository of tuples that can be accessed concurrently
  - Need the senders and receivers to be able to reliably access the correct data
  - Goal: Minimize blocking communication

- Facilitated by the Tuple Server
  - The tuple server listens for and intercepts all MPI_Send/Recv calls
  - Flags are used by the sender to let the tuple server know how to respond to the request
  - Data is stored in a RB-Tree of arbitrary size. Each individual node has a “data tag” (hash)
  - Within each node, there is a queue of messages to be read
  - Data profile: [data tag (hash) | data size | data]

Tuple Space Communication

- Requirements:
  - Non-blocking -- Sender does not wait for message to be received
  - Asynchronous -- Tasks may finish at different rates. It is prefered to maximize time spent computing and minimize time spent waiting. Checkpointing is a good compromise between asynchrony, network/cpu load, and ease of design/use.

```
|---Axxxxx--|     |-------Dxxx--|                   |---A--||-----D|
|--------Bxx-|    |----Exxxx----|         VS        |------B-||----E|              x = wait time for blocking I/O
|------------C|   |-F------------|                  |------------C||-F-|
```

  - Reliability -- Scientific studies require repeatability. Asynchronous communication can lead to race conditions

Tuple Space Communication

- Possible bugs
  - The communication functions and server receive functions use blocking send/recvs. Does this act as a mutex -- if so, this is not truly asynchronous! If not, is it a bug that prevents full performance?
  - Only a single tuple server can currently be used at a time
    - Would it make sense to use more than one? Can we control where it runs?
  - General inefficiencies and bugs scattered about in code. Proper code audit should be completed once current goals are met.

To Do:

- Develop thorough test cases to determine if the current send/recv calls in the put and get functions act as a mutex, benchmark the current performance of the server, then determine if an alternative implementation of communication functions will provide increased performance.

- Determine if multiple instances of the tuple server can/should be run
  - What happens when more than one tuple server is instantiated?
  - Will having more than one instance of the tuple server net any performance benefits?
  - How much of the existing code will have to be updated to allow for this change?

To Do:

- Implement a proper hash function
    - Should be modular as different data has different access requirements and properties. May not be possible to achieve determinism and uniformity over a defined range for all sets of problems with a single hash function.

- Document!!
    - New additions to the code base
    - Mark deprecated and in progress code as well as known bugs
    - Create a detailed roadmap for the project going forward

GUI and Workflow

- Interface is used to organize modules, sets, and groups into “workflow.cfg”

```
num_shared_bc=2000
tuple_space_size=1
modules=(
 {
  function="ielTupleServer";
  args=();
  libtype="static";
  library="libIELexec.a";
  size=1;
 },
 {
  function="laplace0"
  args=()
  libtype="static"
  shared_bc_read=([500,999])
  shared_bc_write=([0,499])
  size=1
 },
 {
  function="laplace1"
  args=()
  libtype="static"
  shared_bc_read=([0,499],[1500, 2000])
  shared_bc_write=([500,999],[1000, 1499])
  size=1
 },
 {
  function="laplace2"
  args=()
  libtype="static"
  shared_bc_read=([1000,1499])
  shared_bc_write=([1500, 2000])
  size=1
 }
)
```

```
workflow:
{
 tuple_set:
 {
  tuple_group:
  {
   order=("ielTupleServer")
   iterations=1
  }
 }
 set1:
 {
  num_set_runs=1
  group1:
  {
   order=("laplace0")
   iterations=1
  }
  group2:
  {
   order=("laplace1")
   iterations=1
  }
  group3:
  {
   order=("laplace2")
   iterations=1
  }
 }
}
```

Acknowledgements

- Funding
  - The National Science Foundation (NSF)
- Facilities
  - The University of Tennessee (UTK) & The Joint Institute for Computational Sciences (JICS)
- Program director/Mentor
  - Dr. Kwai Wong