Scaling and Optimizing Stochastic Tuple-Space Communication in the Distributive Interoperable Executive Library

Zaire Ali (Morehouse College)
Jason Coan (Maryville College)
Mentors: Kwai Wong (UTK) and David White (Maryville College)

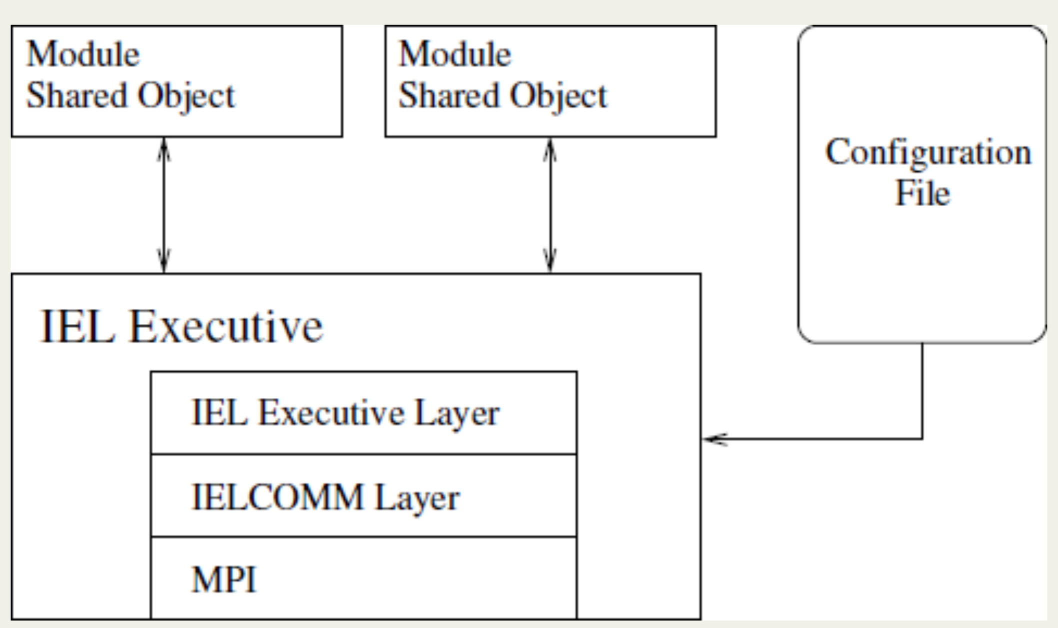
JICS Joint Institute for Computational Sciences
NSF
Oak Ridge National Laboratory
The University of Tennessee Knoxville

Abstract

The Distributive Interoperable Executive Library (DIEL) is a multi-component software framework designed to configure and execute an arbitrary series of intercommunicating parallel physics solvers. The DIEL is capable of running many existing users' codes on high performance computing machines such as Darter and Beacon while utilizing an inter-process interface defined by the user in a configuration file. In addition, it currently provides direct data exchange using user-defined boundary conditions and a prototype of indirect data exchange via a global tuple space. Our task is to improve and extend the tuple space implementation to make it a viable and efficient method of communication.

The DIEL

- Consists of the "Executive" and a communication library
- Executive reads a simple configuration file to execute desired modules and define the shared boundary points between them
- Communication library consists of two parts:
  - **Direct communication** – wrappers for MPI_Send() and MPI_Recv() that enforce shared boundary conditions
  - **Indirect communication** – global "tuple space" used to store data until it is needed



- **Direct communication has the disadvantage of being synchronous, meaning both the sending and receiving processes must be ready at the same time. If the receiver is not ready, the sender must wait. In a system were performance is a key requirement, asynchronous communication should be made available for when processes are not guaranteed to be in sync with one another.**

Existing Prototype of Tuple Space Communication

- A dedicated server function with its own tuple space continuously runs in the background (started by the executive)
- Each committed tuple is associated with a tag, and the server stores and retrieves it according to this tag.
- Memory is dynamically written to a linked list.
- Advantages
  - **Asynchronous, stochastic communication**
  - Allows for a dedicated process to handle communication and memory management
- **But there are problems with the current implementation:**
  - One server process can only handle one request at a time.
  - The code that existed when we arrived was not thread-safe.
  - The existing executive functions and communication library have certain pitfalls that prevent the starting of multiple tuple servers.

Our Development

We decided that our improvement of the DIEL and tuple space should take place in several phases:

1. We expanded the DIEL to accept all C, Fortran, and JAVA based code
   - Developed scripts that accept serial C, Fortran, and JAVA code as input and produce DIEL-infused C code.
   - Developed scripts that execute DIEL module code multiple times across processors simultaneously
2. We allowed for multiple concurrent tuple space servers to be started at the same time.
   - Converted the tuple space into a DIEL module, like any other.
   - Modified the executive to start the number of servers specified in the configuration file. This is done by modifying the calls to **libconfig:**

```
int tupleSize;
if(!config_lookup_int(&cfg, "tuple_space_size", &tupleSize)) {
    ERRPRINTF("\tuple_space_size\" option not set\n");
    cleanup_before_err_ret();
    config_destroy(&cfg);
    return -6;
}
exec_info->tuple_size = tupleSize;
```

   - In order to test this, we modified the IEL_tput and IEL_tget functions to be able to specify a specific server to which to send/receive the data. For example, our new IEL_tput function looks like this **(safety checks omitted for brevity)**:

```
int IEL_tput(size_t size, int tag, int serverRank, void* data) {
    //blocking send of handshake containing size of data to be sent
    MPI_Send(&size, 1, MPI_UNSIGNED, serverRank, TUPLE_PUT, IEL_Comm);

    //blocking send of tuple with the tag by which it will be indexed
    MPI_Send(data, size, MPI_BYTE, serverRank, tag, IEL_Comm);

    return IEL_SUCCESS;
}
```

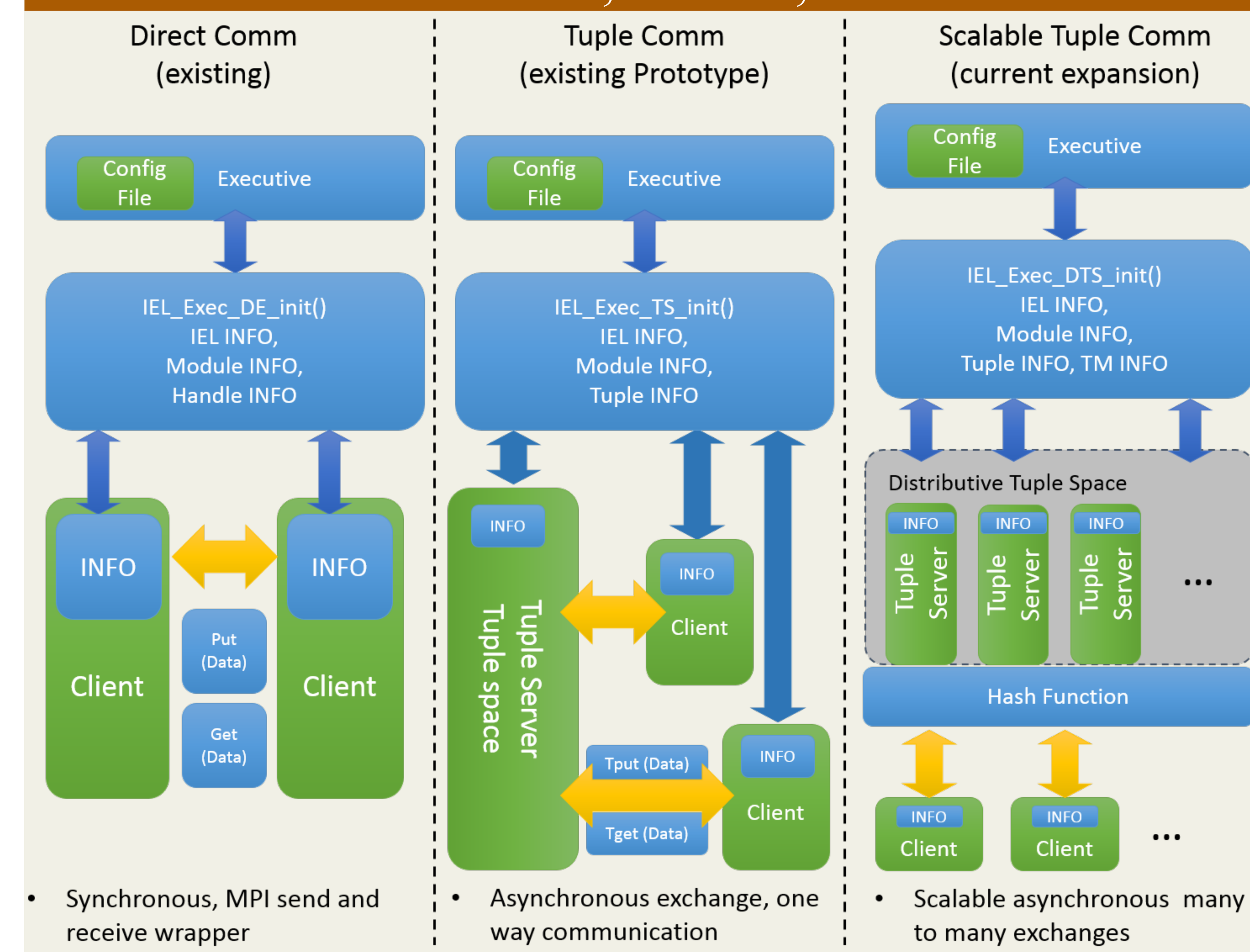
3. We implemented true associativity in the tuple space.
   - The tuple servers use the same shared boundary conditions from the configuration file to index their data, as opposed to an arbitrary tag.
   - This means the tuple servers need to receive the same component information that the executive broadcasts to the other modules.
4. We are currently working on converting the tuple space to a distributed hash table.
   - The hash function enforces the rules of associativity.
   - The hash function should tell modules to which tuple server to send a specific tuple based on the shared boundary condition that it represents.
   - The load of managing the tuple space should be spread as evenly as possible amongst all the tuple servers
   - Once all of this is done, we can once again remove the serverRank parameter from IEL_tput and IEL_tget. Both functions need only to know the shared boundary condition (tag) that the data represents. They can then call the hash function to discover the destination tuple server.

```
int IEL_tput(size_t size, void* data, int sbc) {
    int serverRank = hashFunction(sbc);
    //blocking send of handshake containing size of data to be sent
    MPI_Send(&size, 1, MPI_UNSIGNED, serverRank, TUPLE_PUT, IEL_Comm);

    //blocking send of tuple with the tag by which it will be indexed
    MPI_Send(data, size, MPI_BYTE, serverRank, sbc, IEL_Comm);

    return IEL_SUCCESS;
}
```

The DIEL: Past, Present, and Future



Testing

- We wrote a randomized stress test designed to create many challenging situations for the tuple server algorithm to see how well it handles them.
- Due to the randomized nature of the test, we should run it many times and then look at the distribution of completion times.
- 16 tuple servers, 256 module processes on Darter
- After 40 trials, the tuple servers collectively fulfill an average of **9.6 million tget/tput requests per trial.**
- It takes an average of **7.5 seconds** to complete one trial.
- There is little variation in the time it takes to complete the test, and copious safety checks ensure us that every request is being fulfilled correctly.

Future Goals

While the long-term goals of the DIEL in-general are outside the scope of this poster, we can name the next steps of tuple-space development specifically:

1. The hash function needs to be improved to provide an index on the server as well as the server's rank.
2. Each shared boundary condition represented in the tuple space should have its own dedicated underlying data structure that can act as a queue, stack, generic set, etc. to provide more options for user code.
3. To match functionality of most mature tuple space implementations, we should provide both blocking and non-blocking versions of IEL_tget. Our current function is completely blocking: once it requests a tuple from a server, it will wait until it receives it, even if the producing module has not put the tuple to the server yet.

Contact Information

Zaire Ali, zaireali493@yahoo.com
Jason Coan, jason.coan@my.maryvillecollege.edu
Kwai Wong, kwong@utk.edu