Distributive Interoperable Executive Library (DIEL) for Systems of Multiphysics Simulation

Kwai Wong and Logan Brown
Joint Institute for Computational Sciences
University of Tennessee
Knoxville, TN 37996, USA
kwong@utk.edu
lbrown60@vols.utk.edu

Jason Coan and David White
Department of Computer Science
Maryville College
Maryville, TN 37996, USA
david.white@my.maryvillecollege.edu
Jason.coan@my.maryvillecollege.edu

***Abstract*— As HPC capability and software adaptability continues to expand, the interest to perform complex system-wide simulations involving multiple interacting components grows. In this paper, we present a novel integrative software platform – the Distributive Interoperable Executive Library (DIEL) - to facilitate the collaboration, exploration, and execution of multiphysics modeling projects suited for a diversified research community on emergent large-scale parallel computing platforms. It does so by providing a managing executive, a layer of numerical libraries, a number of commonly used physics modules, and two set of native communication protocols. DIEL allows users to plug in their individual modules, prescribe the interactions between those modules, and schedule communications between them. The DIEL framework is designed to be applicable for preliminary concept design, sensitivity prototyping, and productive simulation of a complex system.**

Keywords—Workflow; Framework; Parallel Computing; Multiphysics Simulation

I. INTRODUCTION

In 1999, the fastest computer in the world could barely reach one TeraFLOPS. Then came the Earth-Simulator, which dominated the Top500 [1] list for the next three years, performing at 33 TeraFLOPS. In just a decade, the number one computer in the 2013 top500 list reached a performance of 33 PetaFLOPS, a whopping 1000 fold of performance increase! More importantly, TFLOPS-scale computers are now considered as commodity machines and are commonly available in the US. Fueled by such a leaping gain in computational capability, the last decade also represents a golden era of physics-based software renaissance. Recognizing the ability to perform complex multiphysics simulations on parallel computers, many software tools have been built to manage the interactions dictated by a mixture of independent scientific components.

In concert with the technological renovation of computer architectures there also appeared the maturity of a number of numerical algorithms and packages in solving large systems of dense and sparse equations on petascale supercomputers. These packages and toolsets such as ScaLAPACK [2], PETSc [3], and Trilinos [4] have been the backbones of many scientific software developments. Taking advantage of object oriented abstractions, equation-based problem solving environments capable to encapsulate mathematical operators in differential or integral equations, such as Diffpack [5], OVERTURE [6], and DEALII [7] were developed to solve PDE systems using finite difference, finite volume, and finite element methods.

As software adaptability for single-purpose application codes continues to grow, to combine and reuse these individual proven software units applied to system-wide applications becomes apparent. The climate community has been the leading authority in its development of CCSM and later CESM [8] component-based climate models. CCA [9] was an effort to sew together different software components for large-scale system-wide simulations on HPC platforms. To anticipate an exascale computer by the end of this decade, the US DOE has increased its investment in software framework infrastructure critical to the success of building next generation energy sources. The Multiphysics Object Oriented Simulation Environment framework, MOOSE [10], provides a software platform to integrate a suite of independent application codes for nuclear reactor simulations. To improve the convergent properties of loosely interacting physics code executing under a unified framework, CouPE [11] (Coupled Physics Environment) is built with a number of coupling schemes to enhance the connection between individual physics modules via interfaces to MOAB [12] and PETSc.

Following the rise of domain specific computational frameworks, a number of open source scientific workflow engines also emerged. The Pegasus project [23] has a large pool of tools that enables scientists to construct workflows for their simulations in abstract terms without worrying about the details of the underlying execution environment on multiple distributed resources. Swift [24] is designed to let users specify, execute, and manage their large-scale science and engineering simulations rapidly through a simple scripting language. It supports applications that execute many tasks concurrently, particularly suited for analyzing large quantities of data in ensemble simulations.

The Distributive Interoperable Executive Library (DIEL) is a lightweight software framework for simulating system-wide scientific applications by combining different interoperable computational components. Similar to many existing large scale multiphysics frameworks developed under DOE initiatives, DIEL is capable of admitting and running many existing users' codes on high performance computers. To sew these codes together, DIEL provides two unique protocols for communication: a direct space exchange unit (DSU) and a tuple space exchange unit (TSU). These units are designed to facilitate transferring and storing data across the memory space on a large-scale supercomputer, depending on the nature of the simulation using DSU for deterministic type of exchanges or TSU for stochastic type of exchanges.

DIEL shares a few similar features of Swift and Pegasus, primarily in the management of the execution and data movements of layers of modular software components. This framework, however, is tailored towards conducting multi-disciplinary research on supercomputers, providing a unified cradle to attain scalable computational performance. It is built with the capacity to collect and assimilate data warranted for new scientific frontiers, resulting in shorter, smarter process cycles and significant cost savings.

The DIEL framework is designed to be an integrated, community-driven, non-proprietary capability that follows an agile development process allowing users to efficiently validate, verify, maintain, and expand new computational models, algorithms, analytics, and tools. Fig. 1 shows a functional block diagram of DIEL. The managing core (EXECUTIVE) and the communication units (COMMLIB) are the critical building blocks of DIEL. These are interconnected with an integrator of numerical libraries and an interface for tools. An analyzer module performs data analytics, sensitivity analysis, and parametric optimization during the life cycle of a simulation. The multiphysics models are distinct computer codes implemented by users to perform their simulations. By leveraging this framework, a biomedical heart-lung simulation is built to showcase the functionalities and the interacting workflow managed in DIEL.

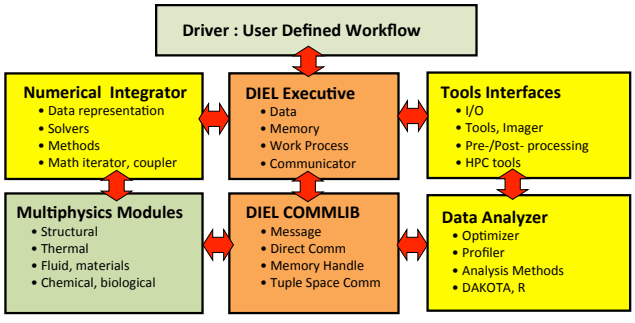

Fig. 1: Functional organization blocks of DIEL

II. Organization of DIEL

A. Design

The development and initial conception of DIEL began in 2010, starting as a software framework to organize the workflow for thermal fluid flow problems. The principal design criteria are two fold - to allow many units of parallel codes running seamlessly under a unified executable and to allow these individual programs to exchange data specified by the user; and a simple efficient engine aimed to utilize the capabilities and resources of large scale supercomputers. The resulting product is a lightweight interoperable executive library built on top of the MPI library. The global MPI_COMM_WORLD communicator is split into many sub-communicators and scattered across separate units of physics codes (modules). A wrapper for the direct communication library, DSU, is built to exchange information between these multiple physics modules.

Fig. 2 illustrates the DIEL software framework. It executes and schedules, in parallel, a series of solvers or modules.
The DIEL is composed of three major components: the Configuration File, the Communicator Library, and the Executive. The Driver, shown in Fig. 3, defines the workflow and functionality of a user's simulation. The second component is the Communicator Library (COMMLIB), which is built as a wrapper for the Message Passing Interface (MPI) and handles the transfer of the data on the shared boundaries between modules. The third component is the Executive, which schedules and manages the workflow of a set of tasks prescribed in the configuration file. A user-specific Driver program initiates the computation by passing a Configuration File to the executive, which organizes and runs the sequence of a simulation. The DIEL can run multiple parallel computational tasks concurrently or in sequence on large scale computing platforms such as Darter (Cray XC30) and Kraken (Cray XT5) at the National Institute for Computational Sciences (NICS).

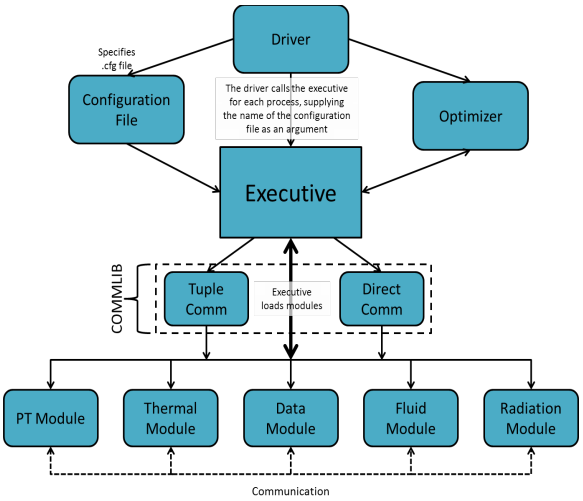

Fig. 2: Design of DIEL

B. *Workflow of DIEL*

The Driver is the unit scheduling the simulation and provides the means for the user to manage the sequences of a complex system-wide simulation. The driver works by loading modules as external libraries and initiating execution using the functions provided by the Executive. The Executive arranges and defines array structures for the data given in the configuration file. The Executive then takes control of the assigned tasks, executes them accordingly, and coordinates the communication among them through the Communicator Library. An example of the workflow and execution of a simple driver program is illustrated in Fig. 3. In this figure a parallel simulation is initialized, then two modules generate multiple combinations of modules and configuration files. Execution of a simulation does not have to occur in a parallel fashion, but can instead be integrated inside control structures, such as loops or if statements, providing control over parameters and input data.

The user-defined configuration file, shown in Fig. 4, is passed into the Executive at runtime and provides all of the necessary

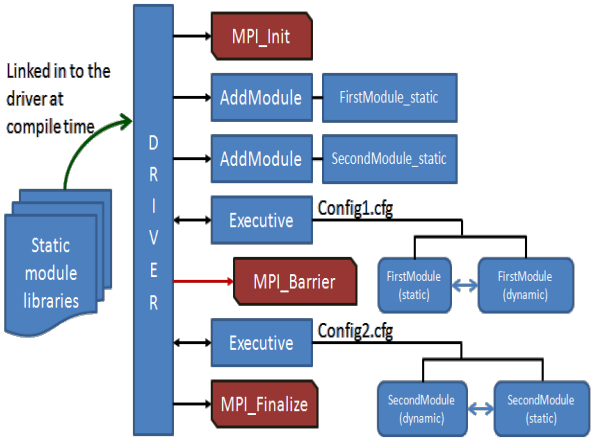

Fig. 3: Typical driver code of DIEL

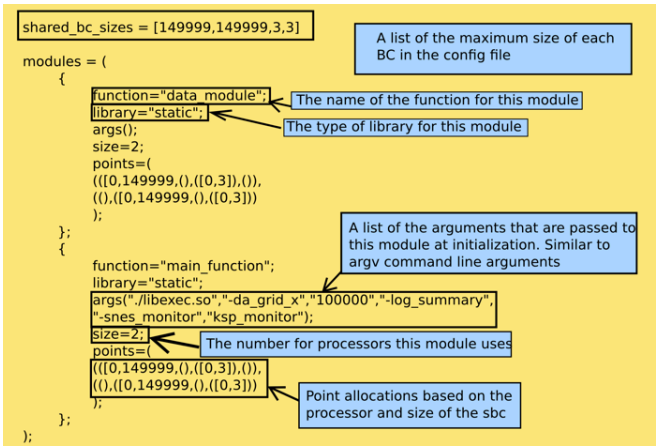

Fig. 4: Example of a configuration file of DIEL

information for the simulation. One of the primary data fields provides information on the shared boundary points specifying the size of boundaries shared between modules. Also listed in the configuration file are the input arguments, the number of processes requested, the library type, and the desired per module memory size of the shared boundary data of each module. This allows the Executive to pre-allocate memory and distribute data accordingly. The configuration file provides the means to define the data and tasks for a simulation in an easily readable text-based format. Because the Configuration File and Executive support multiple module execution, the same module can be called many times during one execution with or without shared boundary points. In addition multiple configuration files can also be used. The DIEL package is organized in four major sections: the DIEL core libraries, the drivers, the third-party mathematical and scientific libraries, and the use cases.

A number of numerical libraries are incorporated to facilitate users to build scalable module codes adapted to DIEL. As an example, Trilinos is extensively used in the solver for the following use case simulation. The suites of dense linear solver, particularly ScaLAPACK and MAGMA [13], developed by the ICL group at the University of Tennessee, are available. To facilitate code migration, a unified interface is developed to encapsulate some of the typical functions available on both GPUs and the Intel many-core coprocessors.

C. *Communication Interfaces*

The coupling between two individual codes is based on the description of their shared array boundaries written in the configuration file. The Configuration File, shown in Fig. 4, defines the functionality of each simulation, the number of shared boundary conditions between different modules, and the number of processors that are assigned for parallel computation. Each module is defined by its name, library type, input data, and a set of shared boundary points. These shared boundary points, assuming they are joined geometrically, are generated by a mesh preprocessing code and written to a single ASCII file.

The COMMLIB is the interface between processes for data communication. After execution of the function *IEL_exec_init()* - reading in the shared boundary conditions and initializing communication data structures containing information about the entire IEL, each specific module, and the data handles specified between modules - the necessary module libraries are loaded and the user-specified communication begins. The client processes are then free to use either of the two communication methods as they see fit for their specific needs.

The first method is direct communication, DSU, facilitated using the shared boundaries specified by the user in the configuration file as gathered by the Executive and the MPI wrapper functions *IEL_put(module info, handle info, data)*

and *IEL_get(module info, handle info, data)*. In those two functions, the user simply passes a description of the module to exchange data, a description of the data to be exchanged, and a buffer to exchange data from or to. The function then finds and verifies the boundary where data is to be exchanged and directly exchanges said data between processes using a non-blocking MPI send (put) and a blocking MPI receive (get) function call. It is necessary that each call for *IEL_put()* be matched by a respective *IEL_get()* on the receiving process, leaving this direct style strictly synchronous. This method is tested on both small-scale machines and large-scale supercomputers such as Kraken and Darter.

The second option of communication is an indirect, cloud-style communication using a tuple space, TSU. In contrast to direct communication, the client is free to exchange data with a shared space using a dedicated communication server, rather than wait on the possibly busy destination process to reach its communication calls or manage the memory itself using one-way MPI function calls. The tuple space code works by using a combination of three functions: a *IEL_tuple_server()*, *IEL_tput(size, tag, data)*, and *IEL_tget()*. An illustration of the current tuple space communication is shown in Figure 5b.

A tuple space is a delocalized collection of tuples that can be committed to access concurrently [14]. It is implemented through a dynamic data structure in memory where users can add to and retrieve from using a certain identification system. Many implementations have been developed in languages such as Java (JavaSpaces) [15] and Linda [14]. A similar style of one-sided delocalized communication exists in Unified Parallel C (UPC) [16] and is available in modern MPI implementations [17]. The DIEL's implementation differs from strictly one-sided communication by using a tuple server to coordinate communication, association, and memory management. The DataSpaces architecture—a programming system for coordinated scientific applications—is a similar implementation of the distributed tuple-space communication layer for interfacing applications [18].

As seen in Fig 6, the *IEL_tuple_server()* function, started currently by the client code running only on rank 0, operates first by initializing a doubly-linked list to serve as a tuple space and then enters a communication loop in which it searches for communication requests or handshakes. When a handshake is found, the function then executes the respective communication with the client process before returning to additional search for handshakes. In the *IEL_tput()* function, the user passes the size of the data to be exchanged, a unique tag to identify the data, and a buffer containing the data to be committed to the tuple space. In the *IEL_tget()* function, the user passes the tag of the data desired from the space, and two buffers to have the data and size written to.

In order to create a fully modular system, the tuple space is converted into a DIEL module like any other. This way, the tuple space can be swapped with a modified version or entirely new implementation without having to recompile the

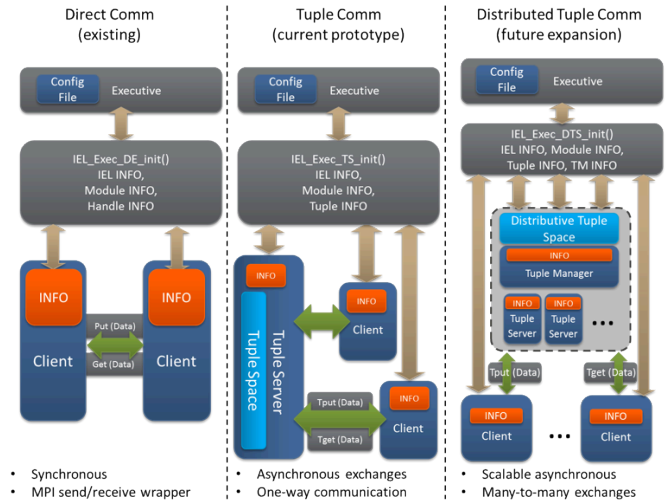

Figure 5: Communication Interfaces of DIEL, left (a), middle (b), right (c).

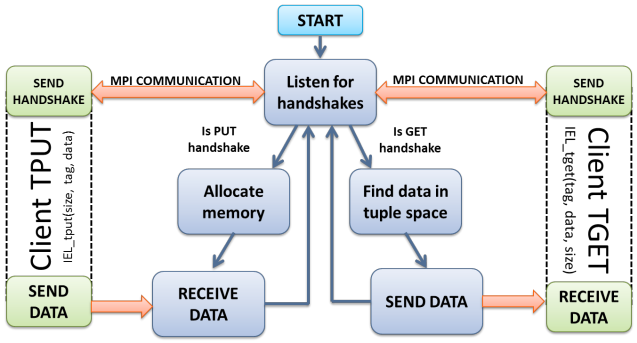

Figure 6: Schematic diagram of tuple space communication in DIEL.

DIEL. Other module processes should be able to use a hash function to discover the proper server for a specific tuple, creating a distributed hash table from the tuple servers. The input to the hash function should be the associativity data for the tuple, which should correspond to the shared boundary conditions defined in the configuration file. This concept of implementation is illustrated in Fig. 5c.

D. Performance Test

The DIEL is capable of running multiple copies of serial or parallel codes concurrently on a large-scale supercomputer. An API is built to assist the transformation of a users' serial or parallel code to conform to the modular format of the DIEL and the generation of the needed configuration file. Fig. 7 shows a weak scaling performance curve of DIEL running multiple instances of the same physics code on Kraken. Each instance of the code uses 32 processes and is treated as an individual physics component (module). The graph compares the average total runtime using DIEL to the runtime of the base code. The performance of DIEL scales well up to 128 instances (4096 processes), when the overhead of DIEL becomes significant.

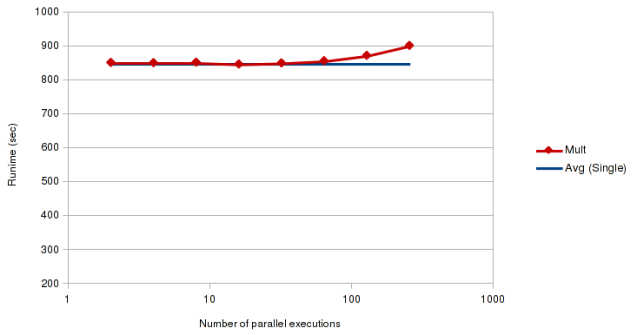

Figure 7: Weak scaling performance of running multiple copies of same physics code.

A randomized test to stress the performance of the TSU is also performed. We ran this test with 16 tuple servers and 256 modules processes on Darter. After 40 trials, the tuple servers collectively fulfill an average of 9.6 million tget/tput requests per trials. Every request is fulfilled correctly. It takes an average of 7.5 seconds to complete, with no obvious outliers.

A fan test is used to compare the performance of the TSU against MPI function calls. In the fan test, process 0 broadcasts *n* different messages, with *n* being the "fan size" or the number of broadcast processes. Two cases are computed, a messages size of 4 Bytes shown in Fig. 8, and 4 Kbytes shown in Fig. 9. For the "MPI_Send/MPI_Recv" and "tput/tget full" cases, process 0 sends these messages to *n* different threads, which then sends back a response message. The total number of messages passed for these cases is twice the fan size times 1,000. For the "tput/tget half" case, the test puts *n* different tuples to the server with *n* calls to *IEL_tput()*, and then gets them all back with *n* calls to *IEL_tget()*. Again, this process is repeated 1,000 times for each run.

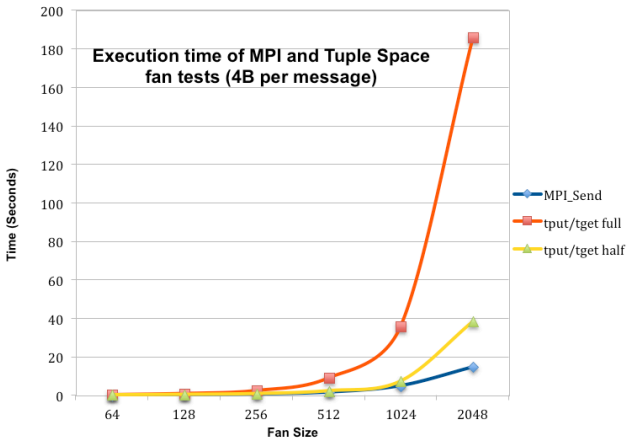

Figure 8: Comparison of performance of MPI and TSU in the fan test.

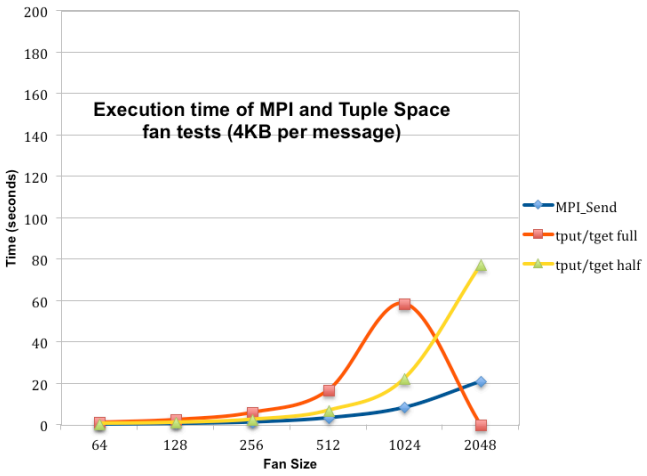

Figure 9: Comparison of performance of MPI and TSU in the fan test.

In Fig. 9, the "tput/tget full" of the fan size of 2048 runs out of memory before completing, indicated by "0" time on the graph. From these tests, exchanging data via TSU is not scalable beyond 256 processes or more appropriate units of concurrent computation. In a system-wide multiphysics simulation, each computational unit may consist of many parallel processes but handled only by a single tuple unit of communication.

III. Use Case Examples

A. *Ising Model*

The Ising Model is a mathematical model used to quantify the energy, magnetization, and heat capacity of atomic particles. In this calculation, the range of temperatures is decomposed evenly and assigned to a process. Each process then uses Markov Chain Monte Carlo methods to approximate the distribution using the replica exchange method [22]. Fig. 10 shows the cost of the Ising model using the TSU to perform millions of small data exchanges.

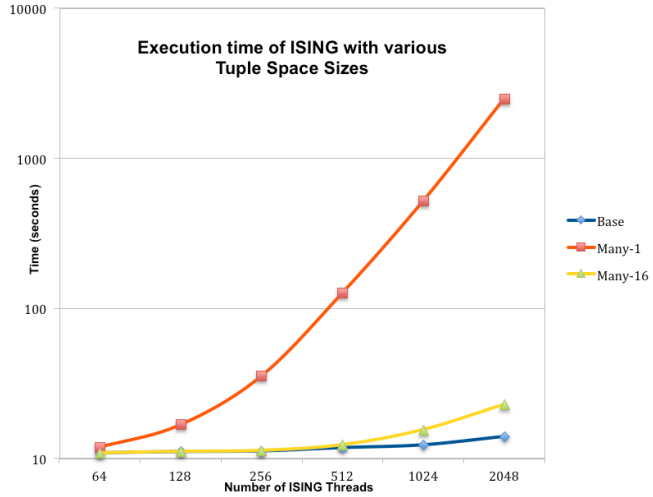

Figure 10: Performance of small data exchanges in the Ising model.

The Base Case is the time it takes when all the exchanges are done with MPI_Send and MPI_Irecv. The Many-1 case is when all the data exchanges are done with a single tuple server. The Many-16 case utilizes 16 tuple servers to perform the same number of exchanges, with messages being put to the servers in a round-robin fashion. The size of the data for each exchange is 3 integers and 7 doubles. The total number of exchanges for each run is 2 times the number of threads times 10,000.

B. *Sphere Packing Model*

The sphere packing code is an example used to demonstrate a typical way to use DIEL. Given a string of P spheres of radius 1, a string of Q spheres of radius 1, and a cylinder of radius R, the goal is to wrap the strings of spheres around the cylinder so that the centers of the final spheres in each string match up perfectly, forming a ring around the cylinder, illustrated in Fig. 11. Given P and Q (the lengths of the strings), the radius of the cylinder and the height difference between the spheres in the string are solved using a genetic algorithm. The code is divided into two loosely coupled DIEL modules. One module takes in sets of parameters and calculates the distance between the final sphere centers (the fitness function). The other module takes in sets of parameters and fitness functions, and creates new sets of parameters based on linear combinations of the fittest ”parents”. These modules communicate through tuple communication, which allows flexibility to perform multiple simulations concurrently. For example, one can run several different sphere packing algorithms with different strings of spheres, and run any number of copies of the genetic algorithm to generate parameters. Fig. 12 shows the workflow where multiple copies of the sphere packing algorithm are computed, but only one copy of the genetic algorithm. The genetic algorithm module receives any population that is ready to be processed, as soon as it has been put in the tuple space. In addition, several identical copies of the sphere packing algorithm can be run concurrently.

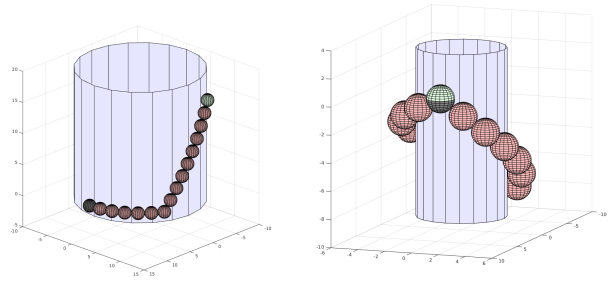

Figure 11:Schmatic diagram of the sphere packing algorithm.

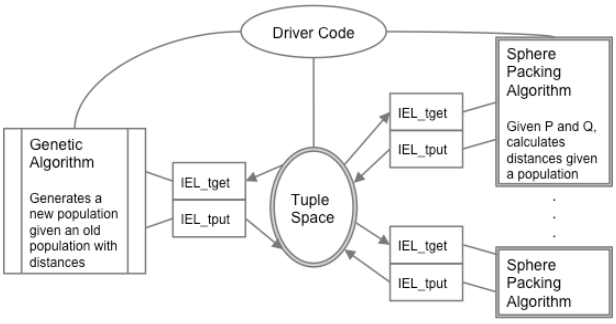

Figure 12:Workflow of the sphere packing simulation.

C. *Cardiac Model*

Cardiovascular disease is the leading cause of death in America. Computer simulation and visualization of the complicated dynamics of the heart have great potential to provide quantitative guidance for diagnosis and treatment of heart problems. There have been intensive research efforts on developing accurate computer models to advance the understanding of the mechanisms of cardiovascular dynamics. To demonstrate the viability and functionalities of DIEL, we have illustrated the workflow of a heart simulation is illustrated in the Fig. 13.

Heartbeats are the result of a sequence of electrochemical excitation waves that are initiated from the sinoatrial node. The electrical impulses induce intracellular calcium cycling, which in turn causes heart muscles to contract. This process, known as excitation-contraction coupling (ECC), is essential to the functioning of a healthy heart. On the other hand, mechanical changes that respond to neural and hormonal influences also impact the electrical properties of the heart. When a part of the heart tissue is stimulated, the induced action-potential propagates out, making the tissue contract. Such action is controlled by a given set of Kirchoff stress parameters and stress activated ion channels. The fluid flow pattern in the deformed ventricle is governed by the Navier Stokes equations and will be solved by an in-house multi-purpose finite element code [19].

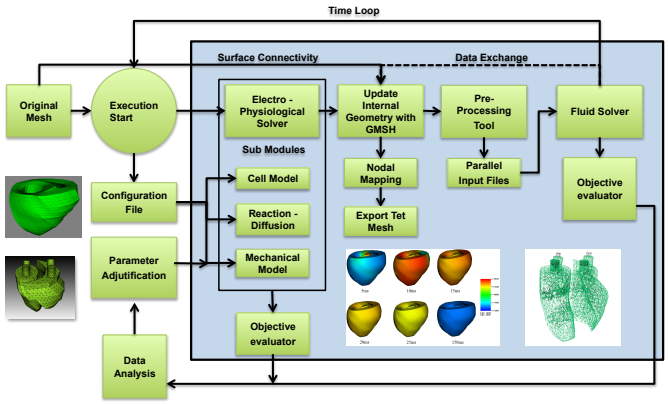

Figure 13: Workflow of a cardiac simulation using DIEL.

The biomechanical module is governed by a reaction–diffusion equation describing the propagating potential and an equilibrium equation describing the stress field [20]. Moreover, the electrical potential involves dozens of ODEs governing variations of various ionic concentrations and gate variables.

The interaction between the biomechanical and fluid modules requires regenerating an internal flow mesh during the course of the simulation. Reconstructing the changed mesh utilizing the Gmsh library [21] is used to reconstruct the new mesh. The Gmsh library is capable of generating internal unstructured tetrahedron meshes based on defined surfaces.

IV. Conclusion and Future Work

The DIEL has given users the ability to run multiple sets of serial and parallel code concurrently on a supercomputer. It allows the management of workflow to be specified in an input file. Two interfaces of communication functions are built to accommodate the transfer of data in deterministic or stochastic manners. The presented DIEL framework is designed to be portable across many computing platforms that are commonly used for simulating many multiphysics problems.

As we work to expand the scope of this framework, we will encompass non-geometrical association written to a binary HDF5 file and XML file that can be taken in by DIEL in parallel. We are also working to extend and improve the performance of current algorithms for tuple space communication. The viability of a mixture of multi-component physics-based code depends on the convergent property of their couplers. We will examine various numerical schemes and leverage the first-hand experiences from the researchers at other institutions to organize a set of coupling techniques in DIEL. Although dynamic runtime task analysis and scheduling can sometimes greatly enhance the performance of a set of loosely connected codes, it will only be considered in some specific cases when the needs from the user community is significant.

Acknowledgment

This material is based upon work performed using computational resources supported by the University of Tennessee and Oak Ridge National Laboratory's Joint Institute for Computational Sciences (http://www.jics.utk.edu). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the University of Tennessee, Oak Ridge National Laboratory, or the Joint Institute for Computational Sciences.

References

[1] www.top500.org

[2] Scalapack : www.netlib.org/scalapack

[3] S. Balay, et al, PETSc Users Manual, Argonne National Laboratory, ANL-95/11 – Revision 3.5, 2014.

[4] Trilonos : trilinos.sandia.gov

[5] H. P. Langtangen, Advanced Topics in Computational Partial Differential Equations - Numerical Methods and Diffpack Programming; Tveito, Aslak (Eds.) Series: Lecture Notes in Computational Science and Engineering, Vol. 33, 2003.

[6] Overture : www.overtureframework.org

[7] DEALII : www.dealii.org

[8] CESM : www2.cesm.ucar.edu

[9] www.cca-forum.org/db/news/documentation/whitepaper05.pdf, High Performance Scientific Component Research: Accomplishments and Future Directions

[10] MOOSE : www.inl.gov/research/moose-applications/

[11] T. J. Tautges, H. Kim, A. Caceres, and R. Jain, "Coupled Multi-Physics simulation frameworks for reactor simulation: A Bottom-Up approach", In International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2011), Rio de Janiero, Brazil, May 2011. American Nuclear Society.

[12] D. Gaston, C. Newman, G. Hansen, and D. Lebrun-Grandi, "MOOSE: a parallel computational framework for coupled systems of nonlinear equations", *Nuclear Engineering and Design*, **239**(10):1768–1778, October 2009.

[13] MAGMA : www.icl.utk.edu/magma

[14] D. Gelernter, "Generative Communication in Linda," ACM Transactions on Programming Languages and Systems, vol. 7, no. 1, pp. 80-112, 1985.

[15] Q. H. Mamoud, (2005, July 12), Getting Started With JavaSpaces Technology: Beyond Conventional Distributed Programming Paradigms, http://www.oracle.com/technetwork/articles/javase/javaspaces, 140665.html

[16] Berkeley Lab, (2013, Nov. 4), *Berkeley Unified Parallel C (UPC) Project*, [Online]. Available: http://upc.lbl.gov/

[17] W. Jiang, et al., "High performance MPI-2 one-sided communication over InfiniBand," Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on. IEEE, 2004.

[18] T. Jin et al., "Using Cross-Layer Adaptations for Dynamic Data Management in Large Scale Coupled Scientific Workflows," ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC), November, 2013.

[19] K. Wong and A. Baker, "A Modular Collaborative Parallel CFD Workbench," J. Supercomputing, vol. 22, pp. 45-53, 2002.

[20] A. Kail, *et al.*, "Interoperable executive library for the simulation of biomedical processes," J. Computational and Applied Mathematics, http://dx/doi.org/10.1016/j.cam.2014.01.011, 2014.

[21] C. Geuzaine and J.-F. cois Remacle, Gmsh reference manual, 2013.

[22] T. Vogel., Y. W. Li,, T. Wust andD. P. Landau, A general parallel framework forWang-Landau sampling. Physical Review Letters 110, 210603, 2013.

[23] E. Deelman, et al, **Pegasus: a Workflow Management System for Science Automation**, *Future Generation Computer Systems*, 2014.

[24] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. vonLaszewski, I. Raicu, T. Stef-Praun and M. Wilde, Swift: Fast, Reliable, Loosely Coupled Parallel Computation IEEE International Workshop on Scientific Workflows 2007.