Constructing a Supercomputing Framework by Using Python for Hybrid Parallelism and GPU Cluster

Yung-Yu Chen* and Sheng-Tao John Yu†
Ohio State University, Columbus, Ohio, 43210

This paper reports the construction of a new software framework, SOLVCON, for high-fidelity solutions of linear and nonlinear hyperbolic partial differential equations (PDEs). Applications of SOLVCON include gas dynamics, aero-acoustics, stress waves in solids, electromagnetic waves, etc. The default numerical scheme employed in SOLVCON is the space-time Conservation Element and Solution Element (CESE) method. SOLVCON is organized based on the two-loop structure found in all finite-volume methods for time-marching solutions: (i) the outer temporal loop for time-marching, and (ii) the inner spatial loops for calculating flux conservation. Built as a software framework that enforces inversion of control, solver kernels in SOLVCON are completely segregated from supportive functionalities and are made pluggable to the framework. As such, SOLVCON is inherently a multi-physics software package. SOLVCON also aims at reducing turnaround time for high-resolution calculations by incorporating hybrid parallelism, i.e., simultaneously using shared- and distributed-memory parallel computing. The code for General-Purpose Graphic Processing Unit (GPGPU) and multi-threaded computing are modularized as optional parts of the solver kernels. Based on the two-loop algorithm, the software structure of SOLVCON has been organized into 5 layers of Python modules. To handle large data sets associated with high-fidelity solutions of PDEs, special modules were developed for parallel input and output (I/O) and in situ visualization. SOLVCON is developed by using Python programming language because it is efficient to be coded and maintained and its unique capability of interfacing with other programming languages for high-performance computing (HPC). By using SOLVCON for typical computational tasks, 99% of the execution time is used in spatial loops, which are implemented by using C with multi-threading computing, and/or CUDA for GPGPU computing. The use of Python does not impair the performance. To model complex geometry, SOLVCON uses unstructured mesh consisting of mixed elements, i.e., triangles and quadrilaterals for two-dimensional problems, and tetrahedra, prisms, and pyramids for three-dimensional problems. SOLVCON has been routinely used for computational tasks running on clusters composed of more than a thousand CPUs. For a supersonic cross flows passing a jet, a benchmarking CFD problem, with 1.3 billion degrees of freedom, SOLVCON delivers the full transient results, including visualized results, in 3 days. SOLVCON has been open-sourced and released (`http://solvcon.net`) under the term of GNU General Public License.

I. Introduction

The standard approach for high-performance computing (HPC) in modern CFD codes has been based on distributed-memory parallel computing by using domain decomposition in conjunction

*Ph.D. Candidate, Department of Mechanical and Aerospace Engineering, `chen.1352@osu.edu`.
†Associate Professor, Department of Mechanical and Aerospace Engineering, `yu.274@osu.edu`.

with Message-Passing Interface (MPI). In recent years, various MPI packages with many advanced features have been developed. To use a computer cluster with each node equipped with multi-core CPUs, the common practice is still to use MPI uniformly for all CPU cores. Aided by advanced MPI libraries, one could achieve HPC without explicitly invoking multi-threading in each computer node by the CFD codes. However, this convenient way of developing codes for HPC cannot be sustained due to ever increasingly complex hardware.

In the coming decade, the driving force of the next-generation HPC is the further development of many-core technologies. Recent results of using General-Purpose Graphic Processing Unit (GPGPU) computing to achieve 10 to 100 times in computational speed-up were indeed remarkable. However, one has to use specialized languages for GPGPU computing, e.g., CUDA, OpenCL, and special C/Fortran libraries for GPUs. Such specialized code must be coupled with the original main codes written by conventional languages, i.e., C, C++, and Fortran, because a GPU needs to be coordinated with the hosting CPU for the computational tasks.

Essentially, in order to use a heterogeneous hardware platform, e.g., a GPU cluster, CFD codes need to be organized to simultaneously use MPI for distributed-memory parallel computing over the networked computer nodes, and special languages, e.g., CUDA and OpenCL, for shared-memory parallel computing in each node equipped with GPUs. Moreover, the many-core technologies are quickly evolving with many new products in the pipeline aiming at combining conventional CPUs with GPUs. It is unclear that the current GPU clusters are the new paradigm of the next-generation hardware platforms for HPC. Code developers are facing the challenging tasks of porting their codes to the evolving heterogeneous platforms.

HPC in the coming decade will be characterized by evolving heterogeneous hardware and supporting software languages and libraries. The tasks of porting the legacy codes with uniform application of MPI to heterogeneous platforms would be laborious and difficult. Without a clear software structure, the resultant codes could be platform-specific and hard to maintain, leading to a short life cycle. Therefore, it is imperative to reorganize CFD codes into a software framework, which requires *inversion of control*[12] so that the code could be easily adapted to a futuristic hardware or software platforms.

In this paper, we report a new software framework, SOLVCON, as a template of the next-generation software framework for CFD codes for HPC by using heterogeneous platforms. The numerical method employed in SOLVCON for solving hyperbolic PDEs is the space-time Conservation Element and Solution Element (CESE) method.[14] The CESE method and SOLVCON use unstructured meshes composed of mixed elements to model complex geometry. The data structure defined in SOLVCON for unstructured meshes dictates all functionalities in the code. As will be shown in the following sections, the data structure in SOLVCON has been carefully designed to facilitate HPC and address the issue of large data set.

User-supplied solver kernels can be segregated from the main software framework and made pluggable to SOLVCON. In other words, the user-supplied solver kernels would specify necessary functionalities tailored to the specific application of interest. In the process, the solver kernels should override parts of the existing code in SOLVCON. While customizable operations are overridden by the solver kernels, supportive functionalities remain in SOLVCON. The entire software framework is unchanged and reusable.

Because the solver kernels are segregated from the main software framework, it can be developed independently by a user. This approach allows systematic development of pluggable modules of different physical processes by implementing specific Jacobian matrices of the hyperbolic PDEs for certain processes of interest. The strategy also results in significant reduction in code development effort for a new solver to simulate a different physical process, because the stationary part of the code remains intact and can be reused. For this reason, the new code is named *Solver Constructor*, or SOLVCON. As such, all supporting functionalities of SOLVCON are repeatedly tested for a

wide range of applications modeled by different hyperbolic PDEs. Consequently, the quality of SOLVCON improves continuously.

The rest of the present paper is organized as following. Section II illustrates the first-order hyperbolic PDEs and the CESE method. Section III reports the structure of SOLVCON and the use of Python scripts. Section IV discusses treatments for large data sets to achieve fast turnaround time for HPC. Section V shows a few examples as code validation cases. SOLVCON's scalability is demonstrated in Section VI. Finally, we draw conclusions in Section VII and list the cite references.

II. Model Equations and Numerical Method

Consider a system of M coupled PDEs formulated in the following vector form:

$$\frac{\partial \mathbf{u}}{\partial t} + \sum_{\mu=1}^{3} \frac{\partial \mathbf{f}^{(\mu)}(\mathbf{u})}{\partial x_\mu} = 0, \tag{1}$$

where $\mathbf{u}$, $\mathbf{f}^{(1)}$, $\mathbf{f}^{(2)}$, and $\mathbf{f}^{(3)}$ are M-component column vectors. For nonlinear equations, $\mathbf{f}^{(\mu)}$ with $\mu = 1, 2, 3$ are functions of $\mathbf{u}$. Equation (1) is in a conservative form, which can be rewritten by using index:

$$\frac{\partial u_m}{\partial t} + \sum_{\mu=1}^{3} \frac{\partial f_m^{(\mu)}}{\partial x_\mu} = 0, \ m = 1, \ldots, M. \tag{2}$$

Aided by the chain rule, Eq. (2) becomes

$$\frac{\partial u_m}{\partial t} + \sum_{\mu=1}^{3} \sum_{l=1}^{M} \frac{\partial f_m^{(\mu)}}{\partial u_l} \frac{\partial u_l}{\partial x_\mu} = 0, \ m = 1, \ldots, M. \tag{3}$$

By considering the term $\partial f_m^{(\mu)}/\partial u_l$ in Eq. (3) as the component of a matrix $A^{(\mu)}$ at mth column and lth row, one can write

$$\frac{\partial \mathbf{u}}{\partial t} + \sum_{\mu=1}^{3} A^{(\mu)} \frac{\partial \mathbf{u}}{\partial x_\mu} = 0, \tag{4}$$

where $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$ are $M \times M$ Jacobian matrices. For nonlinear equations, the Jacobian matrices $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$ are functions of the unknown vector $\mathbf{u}$.

The CESE method[14] is used to solve first-order hyperbolic PDEs, i.e., Eq. (4). The CESE method performs explicit time-marching calculation for the unknowns. The tenet of the CESE method is to treat space and time in an unified way when enforcing space-time flux conservation. An unique feature of the CESE method is separate definitions of Conservation Element (CE) and Solution Element (SE). Over each CE, the space-time flux conservation is imposed. The integration is facilitated by the prescribed discretization inside each SE. The current implementation use a second-order discretization in both space and time. In general, a CE does not coincide with a SE. Compared to the upwind methods, the CESE method does not use a Riemann solver or a reconstruction step as the building blocks. Nevertheless, the CESE method consistently provides solutions of superb quality. The shock capturing capability depends on the weighting functions, unique in the CESE method.[14] Details of the CESE method can be found in.[15,18–21]

III. Pluggable Solver Kernels for Multi-Physics

To solve Eq. (4), the CESE method perform explicit time-marching calculations for the unknowns. For each time step, the solver enforces flux conservation over each CE in the spatial domain. The overall numerical algorithm consists of two sets of mandatory loops: (i) the outer temporal loop for time-marching, and (ii) the inner spatial loops for calculating flux over each CE. Figure 1 depicts the two-loop structure. Almost all execution time for the PDE solvers is used in the spatial loop. In SOLVCON, the temporal loop is embedded in the main framework and remains unchanged. On the other hand, the spatial loops are coded as a part of the solver kernels, which will be supplied by the user.

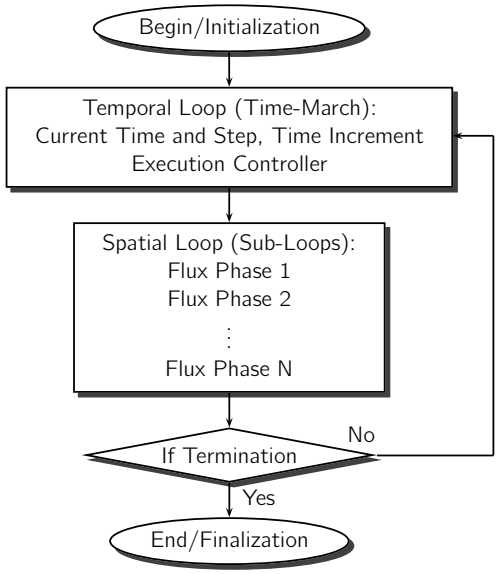


Fig. 1. The two-loop structure (outer temporal loop and inner spatial loops) in the CESE method for time-accurate solutions of the hyperbolic PDEs or conservation laws.

In what follows, we discuss (i) the layered architecture of SOLVCON, (ii) the data structure of unstructured meshes composed of mixed elements, and (iii) the implementation of multiple languages in SOLVCON.

III.A. Layered Architecture of SOLVCON

SOLVCON is highly organized by extensive use of modules, which are written to be reused for all applications. Shown in Fig. 2, the modules and module groups in SOLVCON are organized into a 5-layer structure. In Fig. 2, the top layer is the Python scripts supplied by a user. Lower in the chart near the bottom, the modules are close to the hardware and are not to be changed for all computational tasks. Module `solvcon.mpy` and `solvcon.scuda` are examples. The multi-level modularity of SOLVCON allows user supplied solver kernels to be clearly separated from reusable static code. This is also the key feature for SOLVCON to achieve pluggable multi-physics solver kernels.

The foundation layer provides infrastructure for all functionalities of SOLVCON. Modules in the foundation layer are divided into 4 groups. The *mesh* group defines the data structure of unstructured meshes of mixed elements suitable for domain decomposition. The groups of *distributed parallel* and *shared parallel* contain abstraction to hardware for distributed- and shared-memory parallel computing, respectively. The *utility* group implements other tools for SOLVCON.

The execution layer is the core controller of SOLVCON. It models the two-loop structure of

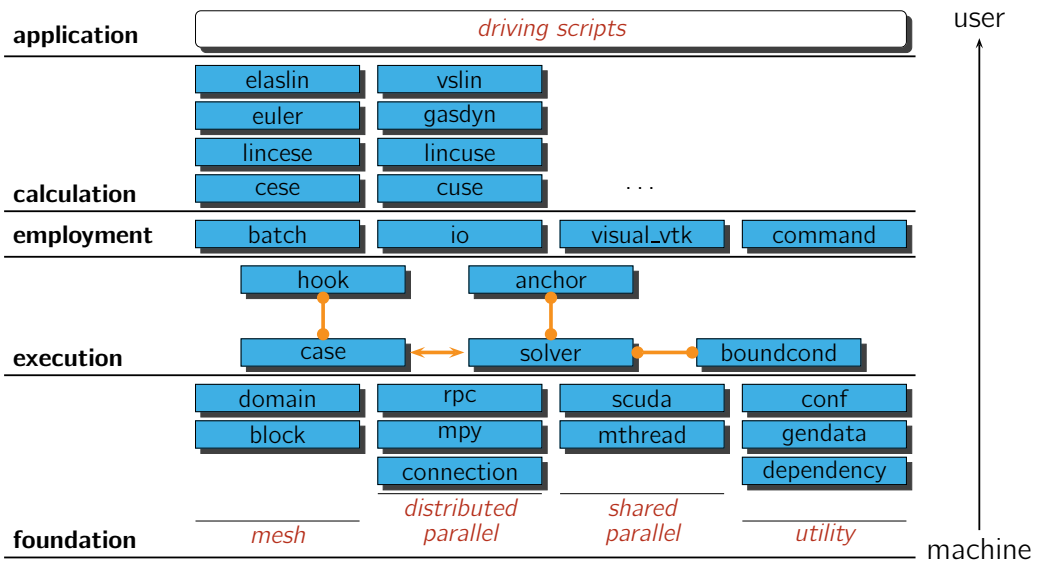


Fig. 2. SOLVCON is organized into 5 layers. Solid boxes are the modules. The double arrow between the modules `case` and `solver` indicates network communication. Italic texts denote sub-systems composed by or using multiple modules.

PDE solvers. The software structure in the execution layer facilitates pluggable physical models in the context of hybrid parallelism. Module `solvcon.case` represents the control logic that operates on the control node for a simulation. Module `solvcon.solver` represents the calculation operations on the compute nodes. The double arrow between the module `solvcon.case` and `solvcon.solver` indicates network communication. Module `solvcon.hook` and `solvcon.anchor` abstracts the supplemental operations on the control and compute nodes, respectively. Module `solvcon.boundcond` is for boundary-condition treatments factored out from `solvcon.solver` for modularity.

Supportive functionalities are in the employment layer. Aided by the abstraction of modules `solvcon.hook` and `solvcon.anchor`, these supportive functionalities are segregated from numerical algorithms, and can be extended by the higher calculation and application layers. There are 4 modules in the employment layer for supportive functionalities. Module `solvcon.batch` is responsible for interoperation to supercomputer batch systems, e.g., Torque. Package `solvcon.io` implements various file formats for input and output (I/O). Module `solvcon.visual_vtk` is a wrapper to interface with the VTK library for in situ visualization. Module `solvcon.command` provides the facilities for command-line interface.

On top of the foundation, execution, and employment layers, the calculation layer provides plug-in modules, including the CESE method and the physical model of interest. Modules in the calculation layer as a group are referred to as the solver kernels of SOLVCON. To date, SOLVCON includes two series of solver kernels. The first is `solvcon.kerpak.cese` series, which implement the CESE method to run on CPU clusters. The second is `solvcon.kerpak.cuse` series, which implement the CESE method for GPU clusters with CUDA. These modules are distributed with SOLVCON in the name space of "`solvcon.kerpak`" for convenience. SOLVCON developers can freely implement custom solver kernels outside the name space.

The application layer is at the highest level. The *driving script* sub-system in this layer allows full customization of the entry point of a simulation. In contrast, conventional simulation codes usually have to hard-wire initialization procedure. Although named as scripts, driving scripts can access all the internals of SOLVCON, and customize everything from the foundation to calculation layer. A driving script can be extended to a fully-fledged Python-based application, utilizing arbitrary functionalities accessible by Python. In this way, the design of user interface is further

separated from the implementation the numerical approach.

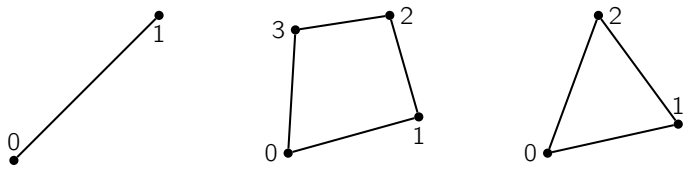
III.B. Unstructured Meshes

SOLVCON uses unstructured meshes[22] of mixed elements to model physical processes related to complex geometry. Two-dimensional meshes can use a mixture of triangles and quadrilaterals, and three-dimensional meshes can mix tetrahedra, hexahedra, prisms, and pyramids. Figures 3 and 4 and Tables 1, 2, 3, and 4 contain essential information for defining the data structure of meshes in SOLVCON. In SOLVCON, a set of arrays serving as lookup tables define the data structure in class `Block` of module `solvcon.block`. These lookup tables are designed to calculate flux conservation based on cell-based meshes. For efficient numerical calculations, PDE solvers directly access the mesh data stored in computer memory. Because the mesh definition is the foundation of numerical algorithm in solving the PDEs, the mesh data structure is placed at the bottom layer of SOLVCON.

The meshes of SOLVCON are composed by a hierarchy of entities including *cells*, *faces*, and *nodes*. SOLVCON assumes the spatial domain is covered by non-overlapping cells. Each cell is enclosed by non-overlapping faces. Each face is formed by non-collocated nodes. A cell in three-dimensional space is a volume, but a surface in two-dimensional space. A face in three-dimensional space is a surface, but a line in two-dimensional space. A node is always a point. Three sets of data are needed to define the mesh entities: (i) the type, (ii) the connectivity, and (iii) the geometry. The type data indicates shapes for each entity. The connectivity data determine the connective relation among the entities. The geometry data store the spatial locations of the entities. SOLVCON defines totally 8 different types of elements, and they are listed in Table 1. Orientation of nodes in the elements are depicted in Figs. 3 and 4.

Table 1. Elements supported by SOLVCON.

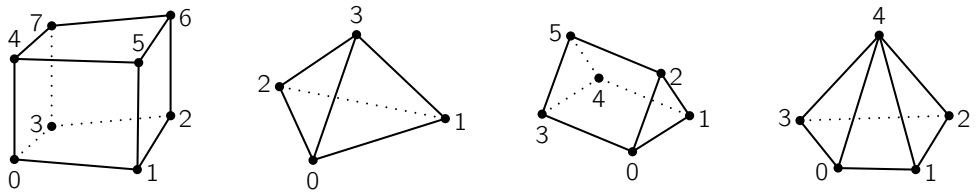
Name	Type	Dimension	Point #	Line #	Surface #
Point	0	0	1	0	0
Line	1	1	2	0	0
Quadrilateral	2	2	4	4	0
Triangle	3	2	3	3	0
Hexahedron	4	3	8	8	6
Tetrahedron	5	3	4	4	4
Prism	6	3	6	9	5
Pyramid	7	3	5	8	5



(a) (b) (c)

Fig. 3. Nodal definition of one- and two-dimensional mesh elements in SOLVCON: (a) Line (type number 1). (b) Quadrilateral (type number 2). (c) Triangle (type number 3).

As shown in Table 1, each type of the elements is composed or enclosed by a set of entities of lower dimension. For example, a triangle contains 3 lines, and a tetrahedron contains 4 triangles.



(a) (b) (c) (d)

Fig. 4. Node definitions of three-dimensional mesh elements in SOLVCON: (a) Hexahedron (type number 4). (b) Tetrahedron (type number 5). (c) Prism (type number 6). (d) Pyramid (type number 7).

Table 2. The relation between a one-dimensional element and its sub-entities.

Shape (type)	Face	= Node
Line (1)	0	0
	1	1

Table 3. The relationship between the two-dimensional elements and their sub-entities. Two-dimensional elements are enclosed by straight lines. Nodal orientation of two-dimensional elements is defined in Fig. 3.

Shape (type)	Face	= Enclosing line formed by nodes
Quadrilateral (2)	0	⁄ 0 1
	1	⁄ 1 2
	2	⁄ 2 3
	3	⁄ 3 0
Triangles (3)	0	⁄ 0 1
	1	⁄ 1 2
	2	⁄ 2 0

Table 4. The relationship between three-dimensional elements and their sub-entities. Three-dimensional elements are enclosed by triangles, quadrilaterals, or a combination of them. □ in the third column denotes quadrilaterals, while △ is for triangles. Nodes in the third column are ordered so that the normal vector of a surface points outward from the volume by following the right-hand rule. Nodal orientation of three-dimensional elements is defined in Fig. 4.

Shape (type)	Face	= Enclosing surface formed by nodes
Hexahedron (4)	0	□ 0 3 2 1
	1	□ 1 2 6 5
	2	□ 4 5 6 7
	3	□ 0 4 7 3
	4	□ 0 1 5 4
	5	□ 2 3 7 6
Tetrahedron (5)	0	△ 0 2 1
	1	△ 0 1 3
	2	△ 0 3 2
	3	△ 1 2 3
Prism (6)	0	△ 0 1 2
	1	△ 3 5 4
	2	□ 0 3 4 1
	3	□ 0 2 5 3
	4	□ 1 4 5 2
Pyramid (7)	0	△ 0 4 3
	1	△ 1 4 0
	2	△ 2 4 1
	3	△ 3 4 2
	4	□ 0 3 2 1

The relations are defined in Tables 2, 3, and 4 for all SOLVCON entities except a point. Two arrays are defined for storing type information of faces and cells:

1. `fctpn`: The array stores type number for each face.

2. `cltpn`: The array stores type number for each group.

Aided by Figs. 3 and 4 and Tables 2, 3, and 4, SOLVCON defines connectivity data for nodes, faces and cells in two- and three-dimensional space. Four two-dimensional arrays are defined:

1. `clnds`: The number of rows is equal to the number of all cells. Each row contains all nodes in a cell. The nodes of a cell are ordered by using Figs. 3 and 4. The first column stores number of nodes for the cell. The first node is stored in the second column. The array holds all nodes in cells. Let N_V denote number of total cells in a mesh $\mathcal{M}$. According to Table 1, the maximum number of nodes in a cell is 8. The shape of array `clnds` must be $(N_V, 1 + 8)$.

2. `clfcs`: The number of rows is equal to the number of all cells. Each row contains all faces in a cell. The faces of a cell are order by using Tables 3 and 4. The first column stores number of faces for the cell. The first face is stored in the second column. The array holds faces in cells. According to Table 1, the maximum number of faces in a cell is 6. The shape of array `clfcs` must be $(N_V, 1 + 6)$.

3. `fcnds`: The number of rows is equal to the number of all faces. Each row contains all nodes in a face. The nodes of a face are order by using Tables 2 and 3. The first column stores number of nodes for the face. The first node is stored in the second column. The array holds faces in cells. Let N_F denote number of total faces in a mesh $\mathcal{M}$. According to Table 1, the maximum number of nodes in a face is 4. The shape of array `fcnds` must be $(N_F, 1 + 4)$.

4. `fccls`: The number of rows is equal to the number of all faces. Each row contains the 2 neighboring cells related by a face. The first column stores the belonging cell, while the second stores the neighboring cell. A face is considered to belong to the belonging cell listed in `fccls`. In geometry arrays, the normal vector of a face always points from a belonging cell to a neighboring cell. The shape of array `fccls` must be $(N_F, 4)$, and the third and fourth columns are used for domain decomposition.

It should be noted that everything in SOLVCON code base is 0-indexed, i.e., the index starts from 0.

Scalars and vectors are defined in geometry data arrays for nodes, faces, and cells. The only geometry array for nodes is `ndcrd`, which is for the coordinate vectors. The number of rows is equal to the number of the total nodes. Each column stores the component of the spatial vector of the nodal coordinates. Let N_D denote the number of total nodes in a mesh M. The shape of array `ndcrd` must be $(N_D, 2)$ or $(N_D, 3)$ in two- or three-dimensional space, respectively. Three arrays are defined for the faces:

1. `fccnd`: The array stores the centers of all faces. The centers are calculated as the centroids. It is a two-dimensional array, and its shape is $(N_F, 2)$ or $(N_F, 3)$.

2. `fcnml`: The array stores unit normal vectors of all faces. It is a two-dimensional array, and its shape is $(N_F, 2)$ or $(N_F, 3)$.

3. `fcara`: The array stores areas of all faces. It is a one-dimensional array, and its shape is (N_F).

Two arrays are defined for the cells:

1. `clcnd`: The array stores centers of all cells. In general, the centers are calculated as centroids of cells. For simplex cases of using only triangles and tetrahedrons for two- and three-dimensional problems, centroids can be replaced by using the in-centers. It is a two-dimensional array, and its shape is $(N_V, 2)$ or $(N_V, 3)$.

2. `clvol`: The array stores volumes of all cells. It is a one-dimensional array, and its shape is (N_V).

Domain decomposition is facilitated by the mesh definition, and the code is organized in module `solvcon.domain`. Invocation of graph partitioner in METIS or SCOTCH library and the splitting algorithm for the mesh data are implemented in the module. To assist manipulation of the data, package `solvcon.io` implements a series of mesh I/O utilities.

III.C. Python for Multi-Language Implementation

Python programming language[23] is used as the building block of SOLVCON. Python has been used for scientific computing and HPC for years.[24–27] In SOLVCON, Python is used to construct the five-layer architecture and achieve modularity. Multiple programming paradigms supported by Python, including imperative, object-oriented, functional, etc., facilitates the task. Python's ability to interface with other programming languages allows the high performance required by SOLVCON.

To use heterogeneous computer architecture and hybrid parallel computing, a software system must accommodate multiple programming languages. For multi-threaded programming, C or Fortran should be used. For GPGPU computing, CUDA or OpenCL should be used. For message-passing, MPI should be used. This is why Python is used as the main language of SOLVCON. By using Python, different programming languages and toolkits can be easily glued together. Python itself serves as a platform to coordinate these computing facilities.

SOLVCON's foundation layer implements the basic facilities that enable multi-language. Module `solvcon.dependency` is designed to help use any binary code that follows C calling convention. The module is based on the `ctypes` module in Python's standard library. Module `solvcon.scuda` is further built upon module `solvcon.dependency` for accessing GPU through CUDA. Aided by the multi-language design, spatial loops can be implemented by using efficient programming languages. Because almost all time in a PDE solver is spent in spatial loops, SOLVCON codes can be as high-performance as legacy codes.

Python is helpful to access other supportive toolkits. To help incorporating MPI and multi-threaded computing, module `solvcon.mpy` and `solvcon.mthread` are developed in the foundation layer. Module `solvcon.visual_vtk` interfaces with VTK library for visualization. Module `batch` wraps a partial interface to Torque batch system. Module `solvcon.io.netcfd` defines a wrapper by using `ctypes` for netCDF library, which is used to access mesh data generated by CUBIT.

Other important benefits are obtained from using Python to construct a software framework: (i) Legacy input files for the PDE solvers are no longer needed. As an interpreted language, Python code are automatically compiled before execution. Driving scripts written in Python will replace the input files. (ii) The dynamic typing system of Python enables high flexibility in controlling the computational tasks. Reflective programming is much easier than writing the conventional codes for instructions. Many functionalities can be implemented freely. (iii) The meta-programming facilities in Python increase the degree of modularity of SOLVCON.

In general, using Python not only increases the capability of SOLVCON, but also reduces the development time. Although the benefits are difficult to be analyzed, it is worth a note that Python results into a compact code base of SOLVCON. SOLVCON's core has less than 10,000 lines

of code, while in-line documentation and comments take 50% of code. A compact code base makes SOLVCON relatively easy to be developed and maintained than legacy codes.

IV. High-Performance Computing

In this section, the implementation of hybrid parallelism for HPC by SOLVCON and the strategy of reducing the turnaround time for HPC tasks are discussed.

IV.A. Hybrid Parallelism

To achieve HPC by using heterogeneous computer hardware in a computer cluster, one has to invoke hybrid parallelism. To control heterogeneous computer platforms, computer programs must be composed by using multiple languages or libraries to coordinate different hardware components. For example, by using GPU cluster, one must invoke shared-memory parallel computing in each GPU for significant speed-up and at the same time use the MPI for distributed-memory parallel computing across the networked nodes. A CPU and a GPU use complete different instructions, and in general they have different machine code. Therefore, the GPU kernel code must be individually compiled and launched from a different set of controlling programs pre-compiled in the hosting CPU. On the other hand, when writing the massively-parallelized code for using a cluster, one has to perform message-passing between networked computer nodes. Thus, one has to perform both share- and distributed-memory parallel computing simultaneously, i.e., the hybrid parallelism.

To achieve hybrid parallelism, SOLVCON framework would take care of the distributed-memory parallel computing by Python and MPI library calls. A user does not need to code any program to control the distributed-memory parallel computing. In SOLVCON, distributed-memory parallel computing is tightly coupled with the two-loop structure. The `BlockCase` class in module `solvcon.case` implements a method containing the temporal loop. The temporal loop controls the parallel process. `Blockolver` objects defined in module `solvcon.solver` are distributed to each of the compute nodes. RPC calls to compute nodes are launched in each iteration of the temporal loop. The `BlockSolver` class contains code for numerical algorithms that calculate the solution on each of the mesh cells, i.e., spatial loops. Data are synchronized among the distributed `BlockSolver` objects through message-passing over network. After a complete execution for a time step, the control is handed back from distributed `BlockSolver` objects to the controlling `BlockCase` object through RPC return.

Shared-memory parallel computing is implemented in segregated solver kernels of SOLVCON. A solver kernel contains a sub-class of `BlockSolver` class, in which spatial loops for a numerical algorithm or a physical model are defined. Shared-memory parallel computing should be used only for spatial loops. Because shared-memory parallelization is confined in the inner-most spatial loops, it does not entangle with distributed-memory parallelization, which is implemented outside spatial loops.

The segregation of parallel paradigms is critically important for maintaining hybrid-parallel codes. The share-memory parallel code in spatial loops can be developed and tested by itself. The spatial loops are interfaced with exterior code through the associated `Solver` class. Because distributed-memory parallel computing is implemented in the structure of `Case` and `Solver` class pair, the two parallel paradigms do not directly couple. Distributed-memory code can also be developed and tested by itself.

IV.B. Reducing Turnaround Time

For high-resolution calculations, large data sets are the bottleneck for slowing down the work flow. Analyzing or visualizing the results can take more time than the computational task. When

calculation is significantly sped up by clusters or GPGPU computing, the bottleneck becomes even more pronounced. Handling the large data and data I/O are indeed the main reason for prolonged turnaround time of all high-resolution simulations. The issue lies in inefficient work flow in the conventional approaches by the legacy codes, in which solution data are output to disks for post-processing and visualizing the results. When the output data are very large, say, in Tera bytes, the post-processing procedure becomes very inefficient. A large storage space is required to store the data, and the I/O process itself could take a lot of time.

In SOLVCON, we first implement parallel data I/O. The code for I/O in the framework is developed to output solutions in parallel from the distributed compute nodes to different hard drives. The solutions are stored in the specific format that can be directly loaded into a parallelized post-processors, e.g., ParaView.

For transient analysis, it is very time-consuming and inefficient to output the full solution and perform post-processing. Sometimes it is impractical because of the large amount of data. For example, to output single-precision data of density, pressure, temperature, velocity, and vorticity, the common variables of fluid dynamics, for 50 million elements, each time step consumes 1.8 GB of storage space. By running 10,000 steps, which are usually minimal for transient analyses, the usage of space becomes 18 TB. Even by reducing the output frequency to every 100 time steps, the data consume 180 GB, which are uneasy to be post-processed. Parallel I/O and post-processing alone cannot solve the issue, and in situ visualization is a must.

To further resolve the issue of the large data set, we have developed the in situ visualization[29] capabilities as an integral part of SOLVCON. Essentially, SOLVCON would call VTK to generate graphics file while the parallelized time-marching calculation is being carried out. As pointed out by Ma,[29] coupling the parallelized numerical algorithms with simultaneous visualization is non-trivial. The execution layer of SOLVCON provides the structure to organize in situ visualization with numerical algorithms. In situ visualization in SOLVCON is a specialized functionality and it is implemented by using the pair of `solvcon.hook` and `solvcon.anchor` modules. The software structure for in situ visualization is exactly the same as that for parallel I/O in SOLVCON. Calling VTK in SOLVCON is straightforward because the VTK package can be directly connected with Python codes. `solvcon.visual_vtk` module provides a thin wrapper from SOLVCON `BlockSolver` objects to VTK objects. The same module also defines the shorthand methods for frequently used visualization pipelines. Similar to that in the parallel I/O functionalities, the in situ visualization is optional to calculations, and can be separately developed and tested.

To recapitulate, SOLVCON supports both (i) parallel I/O and (ii) in situ visualization to address the issue of large data sets. The above two approaches are used simultaneously and they complement each other. The parallelized I/O process still consumes a large amount of storage space for the transient solution, so that it is unsuitable for high-frequency outputs. The in situ visualization process, on the other hand, significantly reduces the size of output data. However, the process could lose details of the solution field if no priori knowledge about the flow field is available. Both techniques are essential for reducing the turnaround time of the HPC tasks.

V. Validation

Many cases of simulation for code validation have been done. These validation cases have been released online with the code. In this section, 2 problems are demonstrated.

A two-dimensional case of Mach reflection is calculated to validate the solver of the Euler equations for inviscid, compressible flows. A mesh composed of 115,608 triangular elements is used for the calculation. As shown in Fig. 5, a normal shock is moving from the left boundary toward the right. The Mach number of the incoming flow is 1.5. For the calculation, the left boundary is treated as a supersonic inlet. The top and bottom boundaries are treated as slip walls. The

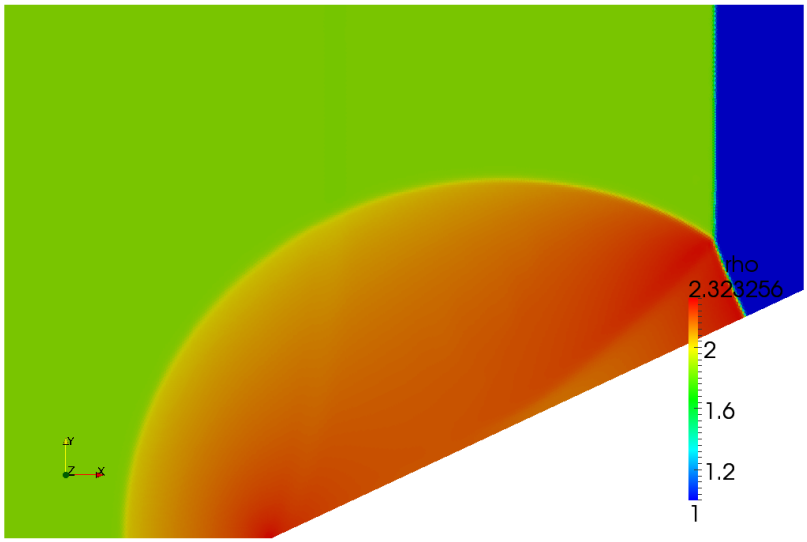


Fig. 5. Snapshots of moving shock reflected by a wedge in a two-dimensional space. The Mach number of the normal shock is 1.5. The specific heat ratio $\gamma = 1.4$, and the maximum CFL number is about 0.93.

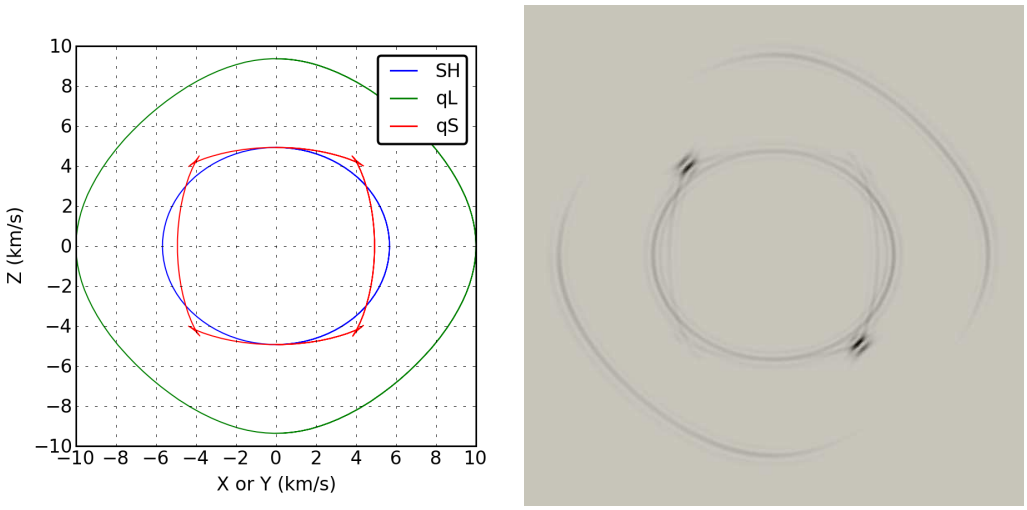
right boundary is treated as a non-reflective boundary. The calculated normal shock, the reflected shock, and the Mach stem are clearly shown in Fig. 5. The results are consistent to the experimental results reported by Schardin.[31]

To demonstrate SOLVCON's pluggable multi-physics, a test case for stress waves in anisotropic is calculated. The calculation has been reported by Yang et al.[34] Stress waves are initiated with a point source at the origin, and expands and moves outward in all directions. Group velocities[33] of the waves are obtained. 250,840 triangular elements are used in the calculation. Figure 6 shows both the analytical and simulated solution for the group velocities of the waves. The simulation calculated the total energy, which is the summation of kinetic energy and potential energy due to stress. The locus of the calculated wave fronts demonstrated that different group velocities occur in different direction. The calculated wave front was validated by comparing to the analytical solution.[32]

VI. Performance Results

To benchmark SOLVCON's scalability, we carried out the simulation of a sonic jet passing a supersonic cross flow. The computational domain is $18D \times 9D \times 9D$ in x-, y-, and z-axes, where D is the diameter of the sonic jet. The Mach number of the supersonic free stream is $M = 1.98$. The density $\rho = 0.86\,\mathrm{kg/m^3}$, and the pressure $p = 41.9\,\mathrm{kPa}$. For the jet, Mach number $M = 1.02$, the density $\rho = 6.64\,\mathrm{kg/m^3}$, and the pressure $p = 476\,\mathrm{kPa}$. The medium is an ideal gas with the specific heat ratio $\gamma = 1.4$.

Figure 7 shows the calculated three-dimensional density contours. The time increment $\Delta t = 7\,\mu\mathrm{s}$, and 12,000 time steps are calculated for the result. 66,791,129 hexagonal elements are used in the unstructured mesh. The leading bow shock and the barrel shock generated by the jet can be clearly seen. Tests are performed with quad-core nodes in the Glenn cluster of the Ohio Supercomputer Center. The interconnect uses 10 Gbps InfiniBand. The calculation shown in Fig. 7 used 66 million cells which include 1.3 billion unknowns. The calculation was performed by using 264 CPU cores and the task was finished in 53 hours. The time includes solution calculation, data output, and in situ visualization. Figure 7 is produced by using the in situ visualization capability of SOLVCON.



(a) (b)

Fig. 6. The group velocities on the 100 plane in a beryl crystal,[32] which is an anisotropic elastic solid of hexagonal symmetry. For elastic solids, the group velocities are equal to the energy velocities.[33] For anisotropic solids, the directions of energy velocities are not always perpendicular to the stress wave fronts, and the cusps shown in the figure occur. (a) shows the analytical solution and (b) plots the contour of energy density calculated form the simulation. The inhomogeneity of the simulated energy density is a result of the anisotropy of the solid. Numerical solution compares well with the analytical solution.

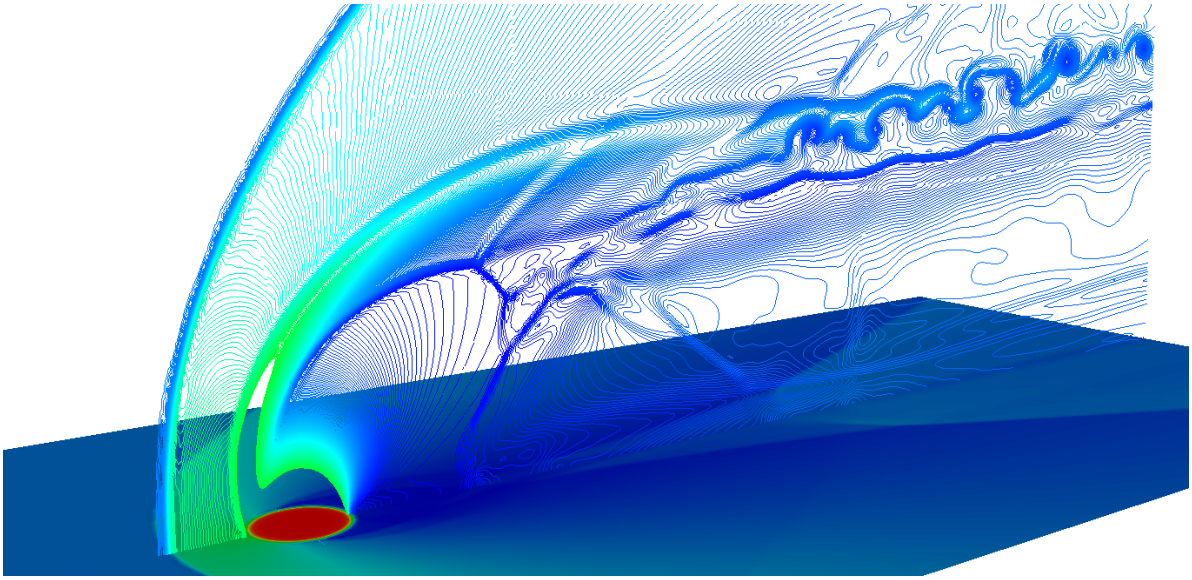


Fig. 7. The calculated density contour of the jet in a supersonic cross flow. The sonic jet transversely enters the supersonic stream from the bottom. The CFD problem solves for time marching solutions of 1.3 billion variables. The computational task was done by using 264 CPU cores. Within 53 hours, the computational task was finished including the time for data output, and in situ visualization.

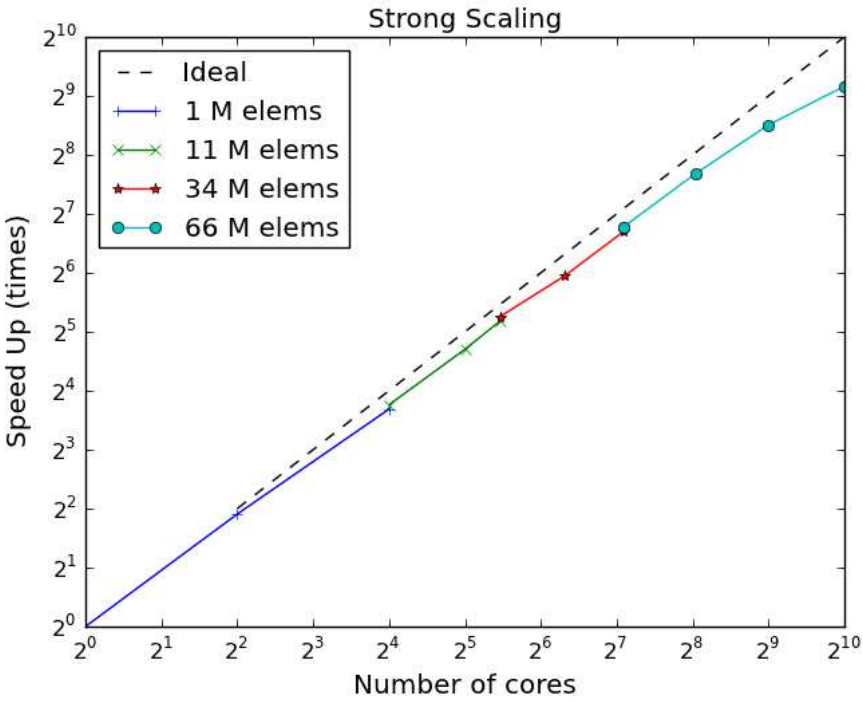


Fig. 8. The speed-up of the strong scaling of SOLVCON. The reference performance is based on the calculation by using 1 million elements and computation by using 1 CPU core. Both x- and y-axes are in logarithmic scale.

Figure 8 shows the strong scaling of SOLVCON. When increasing the number of cores, the problem size is kept constant. The 4 meshes of 1, 11, 34, and 66 million elements are tested. Benchmark results are collect in Figure 8. The x-axis is the number of CPU cores in use, while the y-axis is the speed-up compared to the overall performance of running 1 million element case on 1 core. The figure is in logarithm scale. As shown in Figure 8, the speed-up of SOLVCON is very close to the ideal linear behavior indicated by the dashed line.

There is a mild drop of the speed-up observed in Figure 8 for the case of using 66 million elements. It was caused by excessive network communication. Because the explicit integration in time is required for time-accurate solutions, there must be exchanges of information across the interface among decomposed sub-domains at the end of each time step. The interconnect used in the tests is 10 Gbps InfiniBand, which is much slower than the state-of-the-art interconnect infrastructure that offers 40–80 Gbps. When the number of elements per sub-domain is too low, network communication dominates the simulation time. This is what happens on the 66 million elements cases. The peak overall performance among all tests is 20 Meps, which is obtained by using 1,024 cores for 66 million elements.

VII. Conclusions

In this paper, we report a new software framework, SOLVCON, as a template for the next-generation CFD for HPC by using heterogeneous hardware architecture. SOLVCON is a super-computing software framework that emphasizes inversion of control. SOLVCON employs the space-time CESE method for high-fidelity solutions of generic hyperbolic PDEs and conservation laws. SOLVCON is written by using Python in order to interface with multiple programming languages for HPC and thus it can provide a wide range of diverse supportive functionalities to accommodate complex heterogeneous hardware platforms.

Aided by a highly-organized, layered structure, SOLVCON enables pluggable solver kernels for multi-physics as well as HPC based on hybrid parallelism. To date, the released SOLVCON has included solvers of the Euler equations for inviscid, compressible flows and the velocity-stress equations for stress waves in anisotropic solids. Because of the use of Python, the effort related to code development and maintenance for SOLVCON can be drastically reduced. SOLVCON has been significantly improved since its initial release,[13] including parallel data I/O, in situ visualization, and CUDA-enabled solvers for parallel computing by using Nvidia GPUs. Newly developed functionalities are available for users in building their own solver kernels for specific applications in the future.

SOLVCON has been open-sourced and released under GNU General Public License. Source code and related information can be obtained from `http://solvcon.net/`. New modules for modeling a wide range of physical processes are being developed as a part of continual development of the software framework, including aero-acoustics, waves in viscoelastic materials, and electromagnetic waves. The ultimate goal of SOLVCON is to provide an open platform to general users as a solver constructor, which could be used to maximize the productivity of the code developers and numerical analysts for high-fidelity solutions of generic hyperbolic PDEs and conservation laws.

References

[1]FUN3D, "FUN3D Manual," http://fun3d.larc.nasa.gov/.

[2]Wind-US, "Wind-US Documentation," http://www.grc.nasa.gov/WWW/winddocs/index.html.

[3]NCC, "National Combustion Code Used To Study the Hydrogen Injector Design for Gas Turbines," http://www.grc.nasa.gov/WWW/RT/2004/RT/RTB-iannetti.html.

[4]Morton, S. A., McDaniel, D. R., Sears, D. R., Tillman, B., and Tuckey, T. R., "Kestrel - A Fixed Wing Virtual Aircraft Product of the CREATE Program," Orlando, Florida, Jan. 2009.

[5]Godlewski, E. and Raviart, P., *Numerical Approximation of Hyperbolic Systems of Conservation Laws*, Springer, New York, 1996.

[6]Kulikovskii, A., Pogorelov, N., and Semenov, A. Y., *Mathematical Aspects of Numerical Solution of Hyperbolic Systems*, Chapman and Hall/CRC, 1st ed., Dec. 2000.

[7]LeVeque, R. J., *Finite-Volume Methods for Hyperbolic Problems*, Cambridge texts in applied mathematics, Cambridge University Press, Cambridge, 2002.

[8]Puente, J. d. l., Käser, M., Dumbser, M., and Igel, H., "An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes; IV. Anisotropy," *Geophysical Journal International*, Vol. 169, No. 3, 2007, pp. 1210–1228.

[9]Dumbser, M., Käser, M., and Puente, J. d. l., "Arbitrary high-order finite volume schemes for seismic wave propagation on unstructured meshes in 2D and 3D," *Geophysical Journal International*, Vol. 171, No. 2, 2007, pp. 665–694.

[10]Shi, Y. and Liang, C., "The finite-volume time-domain algorithm using least square method in solving Maxwell's equations," *Journal of Computational Physics*, Vol. 226, No. 2, Oct. 2007, pp. 1444–1457.

[11]Hermeline, F., Layouni, S., and Omnes, P., "A finite volume method for the approximation of Maxwell's equations in two space dimensions on arbitrary meshes," *Journal of Computational Physics*, Vol. 227, No. 22, Nov. 2008, pp. 9365–9388.

[12]Johnson, R. E. and Foote, B., "Designing reusable classes," *Journal of object-oriented programming*, Vol. 1, No. 2, 1988, pp. 22–35.

[13]Chen, Y., Bilyeu, D., Yang, L., and Yu, S. J., "SOLVCON: A Python-Based CFD Software Framework for Hybrid Parallelization," *49th AIAA Aerospace Sciences Meeting, Orlando, Florida, January 4-7, 2011*, Orlando, Florida, Jan. 2011.

[14]Chang, S., "The Method of Space-Time Conservation Element and Solution Element – A New Approach for Solving the Navier-Stokes and Euler Equations," *Journal of Computational Physics*, Vol. 119, No. 2, July 1995, pp. 295–324.

[15]Wang, X. and Chang, S., "A 2D Non-Splitting Unstructured Triangular Mesh Euler Solver Based on the Space-Time Conservation Element and Solution Element Method," *Computational Fluid Dynamics Journal*, Vol. 8, No. 2, 1999, pp. 309–325.

[16]Warming, R. F., Beam, R. M., and Hyett, B. J., "Diagonalization and Simultaneous Symmetrization of the Gas-Dynamic Matrices," *Mathematics of Computation*, Vol. 29, No. 132, Oct. 1975, pp. 1037–1045.

[17]Chen, Y., Yang, L., and Yu, S. J., "Hyperbolicity of Velocity-Stress Equations for Waves in Anisotropic Elastic Solids," *Journal of Elasticity*, Vol. in press, doi: 10.1007/s10659-011-9315-8.

[18]Zhang, Z., Yu, S. T. J., and Chang, S., "A Space-Time Conservation Element and Solution Element Method for Solving the Two- and Three-Dimensional Unsteady Euler Equations Using Quadrilateral and Hexahedral Meshes," *Journal of Computational Physics*, Vol. 175, No. 1, Jan. 2002, pp. 168–199.

[19]Chang, S. and To, W., "A new numerical framework for solving conservation laws: The method of space-time conservation element and solution element," Tech. Rep. E-6403; NAS 1.15:104495; NASA-TM-104495, Aug. 1991.

[20]Chang, S., "On an origin of numerical diffusion: Violation of invariance under space-time inversion," E-7066; NAS 1.15:105776; NASA-TM-105776, July 1992.

[21]Chang, S. C., "On Space-Time Inversion Invariance and its Relation to Non-Dissipatedness of a CESE Core Scheme," *42nd AIAA Joint Propulsion Conference*, July 2006.

[22]Mavriplis, D. J., "Unstructured Grid Techniques," *Annual Review of Fluid Mechanics*, Vol. 29, Jan. 1997.

[23]Rossum, G. v. and Drake, F. L. J., "The Python Standard Library," http://docs.python.org/library/index.html, 2010.

[24]Sanner, M. F., "Python: A Programming Language for Software Integration and Development," *Journal of Molecular Graphics and Modelling*, Vol. 17, No. 1, Feb. 1999, pp. 57–61.

[25]Cai, X., Langtangen, H. P., and Moe, H., "On the performance of the Python programming language for serial and parallel scientific computations," *Sci. Program.*, Vol. 13, No. 1, 2005, pp. 31–56.

[26]Langtangen, H., "A Case Study in High-Performance Mixed-Language Programming," *Applied Parallel Computing. State of the Art in Scientific Computing*, 2008, pp. 36–49.

[27]Langtangen, H. P. and Cai, X., "On the Efficiency of Python for High-Performance Computing: A Case Study Involving Stencil Updates for Partial Differential Equations," *Modeling, Simulation and Optimization of Complex Processes*, 2008, pp. 337–357.

[28]Rossum, G. v. and Drake, F. L. J., "Extending and Embedding the Python Interpreter," http://docs.python.org/extending/index.html, 2010.

[29]Ma, K., "In Situ Visualization at Extreme Scale: Challenges and Opportunities," *IEEE Computer Graphics and Applications*, Vol. 29, Nov. 2009, pp. 14–19.

[30]Chang, S., Wang, X., and Chow, C., "New Developments in the Method of Space-Time Conservation Element and Solution Element – Applications to Two-Dimensional Time-Marching Problems," Technical Report NASA-TM-106758, NASA Lewis Research Center, Dec. 1994.

[31]Van Dyke, M., *An Album of Fluid Motion*, Parabolic Press, Inc., 12th ed., June 2008.

[32]Musgrave, M. J. P., *Crystal Acoustics; Introduction to the Study of Elastic Waves and Vibrations in Crystals*, Holden-day, San Francisco, 1970.

[33]Auld, B. A., *Acoustic Fields and Waves in Solids*, R.E. Krieger, 2nd ed., 1990.

[34]Yang, L., Chen, Y., and Yu, S. J., "Velocity-Stress Equations for Waves in Solids of Hexagonal Symmetry Solved by the Space-Time CESE Method," *ASME Journal of Vibration and Acoustics*, Vol. 133, No. 2, April 2011, pp. 021001.